

Полное справочное руководство

Специальное издание

Использование Visual C++® 6

Кэйт Грегори





Special Edition

USING Visual C++® 6

Kate Gregory



Специальное издание

Использование Visual C++® 6

К. Грегори
(под редакцией Г. П. Петриковца)

ВИЛЬЯМС
Москва • Санкт-Петербург • Киев
1999

ББК 32.973.26-018.2.я75
Г97
УДК 681.3.07

Издательский дом "Вильямс"
Научное редактирование и технические консультации: *Петриковец Г.П.*

По общим вопросам обращайтесь в издательский дом "Вильямс"
по адресу: info@williams.kiev.ua, http://www.williams.kiev.ua

Грегори, Кэйт.
Г97 Использование Visual C++ 6. Специальное издание.: Пер. с англ.— М.; СПб.; К.: Издательский дом "Вильямс", 1999.— 864 с.: ил.— Парал. тит. англ., уч. пос.
ISBN 5-8275-0022-4 (рус.)

В книге широко рассмотрены возможности новейшей версии программного продукта Microsoft Visual C++. Подробно описаны способы применения мастеров, используемых при разработке приложений различного уровня, — с одним документом, многодокументных, с единственным диалоговым окном, с элементами управления ActiveX и модулями DLL. Материал книги дополнен многочисленными демонстрационными программами, в процессе разработки которых максимально используются возможности программных инструментов Microsoft Visual Studio.

Особое внимание уделено новинкам версии 6.0 и новейшим технологиям в программировании приложений, ориентированных на работу в Internet.

Книга рассчитана на широкий круг читателей, интересующихся современными проблемами программирования.

ББК 32.973.26-018.2.я75

ISBN 5-8275-0022-4 (рус.)
ISBN 0-7897-1539-2 (англ.)

Об авторе

Кэйт Грегори (Kate Gregory) — соучредительница консалтинговой компании *Gregory Consulting Limited* (**www.gregcons.com**). Кэйт накопила огромный опыт программирования на C++, увлекшись им задолго до разработки Visual C++ — в незапамятные времена первых компиляторов с этого языка. Фирма *Gregory Consulting Limited* разрабатывает программное обеспечение и страницы для Internet, объединяя возможности современных программных продуктов с потребностями, порожденными этой глобальной сетью, и предлагает на рынке активные Web-страницы, адаптированные к потребностям каждого конкретного заказчика.

Моим детям, Бет и Кевин, которые связывали меня с миром по ту сторону клавиатуры и ежедневно напоминали, как прекрасно ощущение познания нового!

Благодарности

Подготовка книги к изданию — это тяжелая работа. Выполнить ее можно только ощущая постоянную поддержку со стороны. Как всегда, я должна начать с благодарности моим близким за то, что они стоически выдержали все трудности, особенно на заключительном этапе работы над книгой. Брайан, Бет и Кевин, вы держались молодцом! Особенно я хотела бы отметить Брайана, поддерживавшего меня и в качестве заботливого мужа, и в качестве лучшего в мире технического редактора. Мне также посчастливилось работать в тесном общении с Брайаном Оливером (Bryan Oliver), который взял на себя заботы по подготовке иллюстраций, тестированию и отладке программ. Огромное тебе спасибо, Брайан!

Это далеко не первая моя книга в QUE, и среди тех, кто работает в этом издательстве, у меня уже много знакомых, я бы даже сказала, друзей: редакторов, корректоров, художников и других специалистов. Это они превратили мои Word-файлы в книгу, которую вы держите сейчас в руках. Мне очень повезло вновь работать с этой дружной высокопрофессиональной командой. Особенно я хотела бы поблагодарить Олафа Мединга (Olaf Meding) за внимательность и скурпулезность в поиске ошибок и помарок, которые, я сознаюсь в этом, заблудились в моих исходных текстах. Не могу не упомянуть в этой связи Джо Массони (Joe Massoni) и Майка Блажека (Mike Blaszczak), которые также оказали мне громадную помощь в подготовке настоящего издания.

Но если читатели найдут здесь какие-либо огрехи, я прошу отнести их только на мой счет. Пожалуйста, не сочтите за труд прислать свои замечания. Уверяю вас, все они будут учтены мною при подготовке следующих изданий. И, конечно, я не могу не поблагодарить тех читателей, которые прислали отклики на мои предыдущие книги и таким образом содействовали появлению новой.

Введение

Visual C++ 6 представляет собой мощный и сложный инструмент для создания 32-разрядных приложений Windows 95 и Windows NT. Эти приложения намного превосходят как по объему, так и по сложности своих предшественников для 16-разрядной Windows и еще более древние программы, которые вообще обходились без графического интерфейса. Но, несмотря на то что объем и сложность программ увеличиваются, для их создания от программиста требуется не больше, а меньше усилий, по крайней мере для тех, кто правильно выбирает необходимые инструментальные средства.

Именно таким инструментом является Visual C++ 6. Оснащенный набором разнообразных *мастеров* (Wizard), формирующих программный код, этот продукт позволяет в считанные секунды создать вполне работоспособное приложение Windows. Включенная в состав Visual C++ 6 библиотека классов Microsoft Foundation Classes (MFC) уже стала фактически стандартом для разработчиков компиляторов на языке C++. Визуальные средства разработки интерфейса пользователя превращают процесс компоновки разнообразных меню и диалоговых окон в довольно увлекательную игру. Время, которое вы затратите на изучение возможностей этого продукта, сторицей окупится при создании первого же проекта.

Для кого написана эта книга

Эта книга научит вас использовать Visual C++ 6 (для создания 32-разрядных приложений Windows, включая обработку баз данных), Internet и приложения, в которых применяются богатые возможности новейшей технологии ActiveX. Конечно, такое заявление ко многому обязывает, особенно если учесть, что все нужно уместить менее чем в тысячу страниц. Поэтому кое-чего вы здесь точно не найдете. Эта книга *не научит* вас следующему.

- **Языку программирования C++.** Вы уже должны владеть этим языком. По ходу изложения вы только время от времени встретите упоминания о тех или иных соответствующих контексту концепциях *объектно-ориентированного программирования* или конструкциях языка C++. Материал на эту тему — *C++ By Example* (C++ в примерах) — находится на прилагаемом к книге компакт-диске.
- **Работе с приложениями Windows.** Вы должны быть достаточно опытным пользователем Windows — в частности, уметь изменять размеры и положение окон, выполнять двойной щелчок мышью, ориентироваться в пиктограммах на панели инструментов.
- **Использованию Visual C++ в качестве компилятора C.** Если вы уже работали на языке C, то можете, очевидно, пользоваться продуктом Visual C++ как компилятором. А вот новичкам придется освоить данную технологию до изучения этой книги.
- **Программированию для Windows без использования библиотеки MFC.** Прекрасно, если вы с этим знакомы! Но кое-что о составе MFC вы все же узнаете.
- **Детальному описанию методики программирования ActiveX.** На некоторые из таких тем имеются ссылки в главах, касающихся ActiveX. Но это не более чем указание на материал, который вы должны изучить по другим источникам.

Эту книгу следует прочесть тем, кто считает себя принадлежащим одной из следующих категорий.

- Вы кое-что знаете о C++ и методике программирования приложений Windows, но о Visual C ++ знаете лишь понаслышке. Работая с этой книгой, можно изучить продукт Visual C++ значительно быстрее, чем если бы вы просто начали самостоятельно писать программы.
- Вы уже работали с предыдущими версиями Visual C++. Многие пользователи, освоив определенную методику работы, нуждаются в том, чтобы взглянуть на уже, казалось бы,

известные вещи по-новому и попытаться использовать расширенные функциональные возможности, которые предоставляют новые версии инструментальных средств.

- Вы уже какое-то время поработали с Visual C++ и у вас появилось подозрение, что некоторые операции выполняются слишком сложно. Возможно, так оно и есть.
- Вы уже достаточно освоили Visual C++, но теперь хотите расширить возможности создаваемых продуктов. В этой книге вы найдете достаточно распространенные прототипы для решения таких задач, как оснащение продукта оперативной справкой и эффективными методами вывода на печать, а также программирование параллельных процессов.

Прежде чем приступить к чтению...

Прежде чем начать читать эту книгу, вам придется приобрести копию дистрибутива продукта Visual C++ 6.0 и затем установить его на компьютере. Сам по себе процесс установки настолько прост, что мы не сочли нужным загромождать книгу его описанием.

Еще до того как покупать Visual C++ 6.0, нужно установить на компьютере 32-разрядную операционную систему Windows: Windows 95, Windows NT Server или Workstation. Это, в свою очередь, требует достаточно мощного ПК — скажем, процессор должен быть не хуже 486, по крайней мере 16 Мбайт памяти и не менее 500 Мбайт пространства на диске, а также дисплей с разрешением не ниже 800×600 пикселей. Иллюстрации в этой книге сделаны с экрана монитора с разрешением 800×600 пикселей, и, как вы увидите, некоторые из них выглядят слегка перегруженными. Все исходные тексты примеров имеются на Web-сервере, поэтому работать будет значительно легче, если у вас есть модем и доступ к Internet.

И наконец, вы должны пообещать самому себе терпеливо идти вперед в обнимку с Visual C++ — читать эту книгу, щелкать мышью, вводить что-то с клавиатуры и, в конце концов, добиваться своего. Если не хотите, не набирайте тексты программ с клавиатуры — все они есть на Web-сервере. Но, по крайней мере, нужно быть готовым открывать файлы и анализировать тексты программ по мере изучения книги.

Содержимое книги

Такая обширная тема, как программирование для Windows с использованием Visual C++, включает множество вопросов. Эта книга разделена на главы, посвященные собственно обучению (с 1 по 28), и приложения (от А до Е). Просмотрите сейчас внимательно содержание приложений, перечисленных ниже, и по мере изучения материала обращайтесь к ним, если вы не знаете, как правильно выполнить ту или иную процедуру.

- Приложение А. **Обзор языка C++ и основные концепции объектно-ориентированного программирования.** Это приложение напомнит вам об основных идеях объектно-ориентированного программирования, на которых базируется язык C++.
- Приложение Б. **Программирование для Windows и класс CWnd**. Здесь речь идет об особенностях программирования для Windows, которые скрыты сейчас от пользователя такими классами из MFC, как `CWnd`.
- Приложение В. **Интерфейс Visual Studio**. Здесь приведены сведения обо всех меню, панелях инструментов, зонах редактирования на экране, клавишах быстрого вызова и других подобных элементах, которые и составляют этот довольно сложный, но очень развитый интерфейс между разработчиком и Visual Studio.

- Приложение Г. **Отладка**. Здесь описаны меню, окна, панели инструментов и команды, которые имеют отношение к процессу отладки и пробным сеансам выполнения приложения.
- Приложение Д. **Макросы и глобальные объекты MFC**. Здесь перечислены макросы препроцессора, глобальные переменные и функции, которые используются в заготовках текстов и программ, генерируемых мастерами Visual Studio.
- Приложение Е. **Полезные классы**. Здесь перечислены классы, используемые для манипулирования датами, строками и множествами объектов в примерах, которые рассматриваются в этой книге.

Вы можете не читать все главы подряд, изменять порядок изучения отдельных тем, возвращаться к вопросам, пропущенным при первом чтении, и вообще делать с этим материалом все, что вам заблагорассудится (это зависит от предшествующего опыта и индивидуальных наклонностей). Мы, однако, рекомендуем не пренебрегать теми несложными приложениями, которые включены в книгу. Они смогут вас многому научить.

Основной материал изложен в 28 главах. Каждая из них посвящена отдельной, достаточно автономной теме, но в редких случаях некоторые главы тематически связаны. Примером может служить создание панелей инструментов и включение в приложение оперативной справки. В каждой главе даны подробные инструкции по созданию одного или нескольких вполне работоспособных приложений.

В первых девяти главах описаны основные концепции программирования, которые имеют отношение едва ли не ко всем приложениям Windows. После того как вы получите достаточно четкое представление о них, будут рассмотрены более конкретные задачи.

Ниже перечислены основные темы книги.

Элементы управления и средства диалога

Какое приложение Windows обходится без диалогового окна или без текстовых полей, или без кнопок? Диалоговые окна и элементы управления являются жизненно важными средствами интерфейса пользователя. Каждое из них, даже простенькая надпись из нескольких букв, является, в свою очередь, окном. Использование стандартных элементов управления имеет несомненное преимущество, поскольку экономит и время пользователя, которому не нужно заново изучать технологию манипулирования уже знакомыми ему окнами типа File Open или средствами навигации по дереву каталогов, и время разработчика. Узнать больше обо всех этих элементах управления вы сможете в главе 2, *Диалоговые окна и элементы управления*, и главе 10, *Элементы управления общего назначения*.

Сообщения и команды

Сообщения — это основной элемент программирования в среде Windows. Что бы ни случилось в Windows-машине (щелкнет ли пользователь кнопкой мыши, нажмет ли клавишу на клавиатуре), все эти *события* порождают *сообщения*, которые передаются в одно или даже более окон. Те, в свою очередь, с ними что-то делают. Visual C++ облегчает создание программ, которые перехватывают сообщения и обрабатывают их. В главе 3, *Сообщения и команды*, рассматривается концепция формирования и обработки сообщений и то, как MFC и другие компоненты Visual C++ помогают программисту справиться с ними.

Парадигма документ/представление

Парадигма — это модель, способ взглянуть на вещи со стороны. Создатели MFC приняли в качестве базовой концепции следующее предположение: любая программа формирует нечто такое, что предполагает сохранить в виде файла. Именно эту совокупность информации и обозначили как *документ* (document). *Представление* (view) — это один из способов просмотра документа. В таком разделении содержимого и формы есть много преимуществ, которые подробно рассматриваются в главе 4, *Документы и представления*. MFC предлагает набор классов, которые можно наследовать при создании собственных классов документов и классов представлений. В результате такие распространенные задачи программирования, как, например, экранная прокрутка "крупногабаритных" объектов, решаются беспроблемно.

Вывод на экран

Какую бы умную программу вы ни создавали, вам не обойтись без вывода на экран либо текста, либо графики. Без этого вся "заумность" вашей программы останется "вещью в себе", как любили выражаться классики немецкой философии. Значительная часть работы в этой сфере может быть выполнена автоматически классами представления (в этом одно из преимуществ парадигмы *документ/представление*). Но не волнуйтесь: вам придется многое сделать и самому. О том, что такое *контекст устройства* (device context) и прокрутка, а также о многом другом вы узнаете из главы 5, *Вывод на экран*.

Вывод на печать

Включение в разрабатываемое приложение вывода на печать — зачастую самая простая из задач. Дело в том, что если уж вы организовали в программе вывод на экран, то можете тот же код использовать и для вывода на печать. Но если речь идет о выводе более чем одной страницы информации, задача несколько усложняется. В главе 6, *Распечатка и предварительный просмотр*, рассматриваются возникающие при этом проблемы и способы их решения, а также *режимы наложения* (mapping modes), формирование верхних и нижних колонтитулов и многое другое.

Работа с файлами

Некоторые весьма полезные средства, такие как экранный калькулятор и окно для обмена краткими сообщениями в сети, нужны только временно. Большинство же программ сохраняют информацию в файлах, а затем многократно открывают и закрывают их в процессе модификации ранее сохраненных документов. MFC позволяет значительно упростить программирование операций архивирования и расширить возможности операторов вывода в поток (в частности, операторов >> и <<). О программировании операций с файлами вы узнаете из главы 7, *Сохранение-восстановление объектов и работа с файлами*.

Программирование с использованием технологии ActiveX

ActiveX — это усовершенствованная технология *OLE*, которая значительно упрощает взаимодействие приложений на уровне объектов, позволяя, например, ввести документ Word в рабочий лист Excel или выполнить подобные манипуляции сотнями других объектов в приложениях, разработанных в рамках этой технологии (в дальнейшем — *ActiveX-приложениях*). Различным аспектам использования технологии ActiveX посвящен целый ряд глав: глава 13,

Концепции технологии ActiveX, глава 14, *Создание приложения контейнера ActiveX*, глава 15, *Создание приложения сервера ActiveX*, глава 16, *Создание сервера автоматизации*, и глава 17, *Создание элемента управления ActiveX*.

Internet

Microsoft полагает (и небезосновательно), что распределенные вычисления, при которых нагрузка распределяется между двумя или более компьютерами, становятся все более и более обычным делом в мире ПК. Программы должны "разговаривать" друг с другом, пользователи — посылать сообщения по сети (локальной или глобальной), а MFC — обеспечивать всех желающих создавать такого рода программные продукты с помощью соответствующих классов. В этой книге проблемам программирования для Internet посвящены четыре главы: глава 18, *Windows Sockets, MAPI и Internet*, глава 19, *Использование классов WinInet при программировании для Internet*, глава 20, *Создание элемента управления ActiveX для Internet*, и глава 21, *Библиотека Active Template Library*.

Доступ к базам данных

Программирование обработки баз данных также упрощается. Пакет ODBC (Open DataBase Connectivity) от Microsoft позволяет программировать функции API, которые обеспечивают доступ к базам данных в самых разнообразных форматах: Oracle, DBase, рабочие листы Excel, обычный текстовый формат, прежние версии систем для больших компьютеров, использующие SQL, и многие другие. Вам нужно только вызвать стандартную функцию и средства API, которые поставляются разработчиком или дистрибьютором конкретной базы данных и берут на себя все заботы по выполнению преобразования. Подробности, касающиеся этой темы, вы найдете в главе 22, *Доступ к базам данных*, и главе 23, *SQL и редакция Visual C++ Enterprise Edition*.

Новейшие тенденции

Пользователям, которые, как они полагают, уже стали мастерами в применении базовых методов программирования на Visual C++, предлагается несколько глав, в которых собран материал о новейших тенденциях в этой области. С их помощью вы сможете повысить свое мастерство до такого уровня, который доступен только асам программирования. Здесь вы сможете узнать, как избежать проблем с памятью, расширить "узкие" места и найти "занозы" в тексте программы. Методика выполнения подобных операций изложена в главе 24, *Повышение производительности приложений*.

Концепция повторного использования приобретает в последнее время огромную популярность в программировании, особенно среди менеджеров программных проектов, озабоченных проблемой снижения издержек. Если вы хотите освоить создание такого рода программных продуктов, обратитесь к главе 25, *Как достичь повторного использования программных компонентов* , в которой найдете все необходимое.

Язык C++ сравнительно молод и изменяется едва ли не ежегодно. По мере того как Комиссия Американского института стандартов (ANSI — American National Standard Institute) работает над стандартизацией языка в направлении создания очередного окончательного стандарта, изготовители компиляторов включают в свои версии все новые и новые ключевые слова и возможности. Глава 26, *Исключения, шаблоны и последние модификации C++*, познакомит вас с новинками этого языка.

Поскольку требования пользователей к возможностям программных продуктов непрерывно возрастают, программисты просто вынуждены постоянно обновлять методику работы, создавая приложения, которые обеспечивают все более быструю реакцию на события. Для

многих разработчиков создание приложений с параллельным выполнением функций является жизненной необходимостью, вытекающей из потребностей рынка. Об этом рассказано в главе 27, *Многозадачность на основе потоков Windows*.

Глава 28, *Что еще полезно знать*, послужит для вас путеводителем по темам, которые из-за ограниченности объема мы не смогли включить в эту книгу. Это не более чем самое поверхностное изложение таких вопросов, как создание консольных (не соответствующих спецификации API) приложений, разработка динамически связываемых библиотек и работа с *уникодом* (Unicode).

Соглашения, принятые в этой книге

Само собой разумеется, что в этой книге очень часто встречаются фрагменты текстов программ. Иногда это одна-две строки, в которых текст программы перемежается собственно текстом главы, как, например, ниже:

```
int SomeFunction( int x, int y)
{
 return x+y;
}
```

Отличить текст программы от собственно текста книги довольно легко, поскольку мы применяем в этих фрагментах различные шрифты. Иногда фрагменты текста программы довольно объемны и не перемежаются текстом изложения. Именно такой вариант представлен в листинге 1.

Листинг 1. Пример листинга

```
CHostDialog dialog(m_pMainWnd);

   if(dialog.DoModal() == IDOK()
   {
     AppSocket = new CSocket();
     if (AppSocket->Connect(dialog.m_hostname, 119))
     {
        while (AppSocket->GetStatus() == CONNECTING)
        {
           YieldControl();
        }
        if (AppSocet->GetStatus() == CONNECTED)
        {
           CString response = AppSocet->GetLine();
            SocetAvailable = TRUE;
        }
     }
   }
   if (!SocketAvailable)
   {
     AfxMessageBox("Cant connect to server. Please
♥quit.", MB_OK|MB_ICONSTOP);
   }
```

В предпоследней строке листинга вы видите символ ♥, который мы назвали *символом продолжения программной строки*. Он указывает место разрыва программной строки, который пришлось сделать при верстке текста, чтобы уместить программную строку в формат книжной страницы. В действительности в тексте программы обе части разорванной строки слиты воедино. Если вам придется копировать вручную тексты программ из данной книги, то

в этом месте не разрывайте программную строку, а продолжайте вводить текст в той же строке. Если вы воспользуетесь прилагаемым компакт-диском, то при просмотре текстов программ не увидите никакого разрыва строки на этом месте. Поэтому пусть вас не смущает некоторая разница в том, что вы видите на странице книги и на экране дисплея[1].

Главное назначение фрагментов текстов программ в книге — проиллюстрировать те или иные процессы по ходу изложения, а не быть источником для копирования вручную. Все тексты программ есть на Web-сервере. Не пренебрегайте им — это избавит вас не только от затрат времени на ввод вручную, но и от поиска ошибок (а они при вводе будут неизбежны). Иногда по ходу изложения мы приводим несколько вариантов программы, демонстрируя процесс ее создания и совершенствования. На сервере хранится только окончательная версия. Адреса серверов — **www.mcp.com/info** и **www.gregcons.com/uvc6.htm**.



Это совет: здесь мы обращаем ваше внимание на какую-либо интересную подробность или особенность.

На заметку

Это заметка: здесь описано то, на что очень рекомендуется обратить внимание. Не пренебрегайте сведениями, которые выделены таким образом. Это относится и к тем, кто считает излишним терять свое время на выслушивание (или хотя бы чтение) чужих советов.

Внимание!

Это предупреждение, причем мы отнюдь не шутим. Оно предотвратит очень серьезные последствия (естественно, неприятные) ошибочного шага, который вы можете сделать в той или иной ситуации. Так что материал, выделенный подобным образом, не рекомендуется читать вполглаза.

Если вы увидите слово, выделенное *курсивом*, то где-то рядом ищите определение этого термина или понятия. Иногда это также означает, что мы хотим обратить на него ваше внимание. Имена переменных, функций, классов C++ и вообще все, что вам когда-либо придется вводить с клавиатуры в том или ином виде (например, имена файлов или сообщения), мы выделяем в тексте `моноширинным` шрифтом. Адреса в сети Internet (URL — Uniform Resource Locator) выделены **полужирным** шрифтом. Помните, что URL никогда не завершается символом пунктуации. Так что любые занятые или точки после URL, которые вы встретите в книге, относятся к собственно тексту, а не к коду URL.

Итак, приступим...

На этом мы заканчиваем затянувшееся вступление. Все необходимое для того, чтобы приступить к делу, уже сказано, включая некоторые замечания относительно разметки текста в книге. Настало время заняться изучением методов создания приложений Windows на базе MFC, а затем и разработкой своих собственных приложений. В добрый путь! Пусть не покажется он вам слишком ухабистым! Как говорят французы, ***Vouloir c'est pouvoir***: "Хотеть — значит мочь".

[1] Другое отличие текстов программ, приведенных на страницах книги, от текстов, хранящихся на сервере, связано с переводом комментариев на русский язык. — *Прим. ред.*

ЧАСТЬ I

Первые шаги

В этой части...

ГЛАВА 1

Создание первого приложения

В этой главе...

Создание приложения Windows

Создание простого диалогового приложения

Создание динамически связываемых библиотек, консольных приложений и т.п.

Изменение настройки параметров проекта

Текст программы, формируемый AppWizard

Содержимое MDI-приложения

Простое диалоговое приложение

Обзор настроек AppWizard и глав книги

Создание приложения Windows

Visual C++ — это не просто компилятор программного кода; это *генератор* программного кода. С его помощью в считанные минуты можно создавать приложения Windows, "заказав" генератору приложений AppWizard приготовить для вас некоторый комплексный обед из уже готовеньких блюд — фрагментов программного кода. В конце концов, вы ведь не первый программист, которому понадобилось иметь в приложении окно регулируемых размеров с кнопками максимизации и минимизации, а также меню File, в которое включены команды Open, Close, Print Setup, Print и Exit, не так ли?

Мастер создания приложений AppWizard позволяет создавать много разных типов приложений, но первое, что обычно требуется большинству пользователей, — это *выполняемая программа* (файл приложения с расширением .exe). Кроме того, большинство также хотело бы получить от AppWizard готовые фрагменты программного кода — классы, объекты, функции, которые присутствуют едва ли не в каждой порядочной программе.[2] Для того чтобы создать программу подобного типа, выберите File⇨New, а затем — вкладку Projects в окне New, как это показано на рис. 1.1.

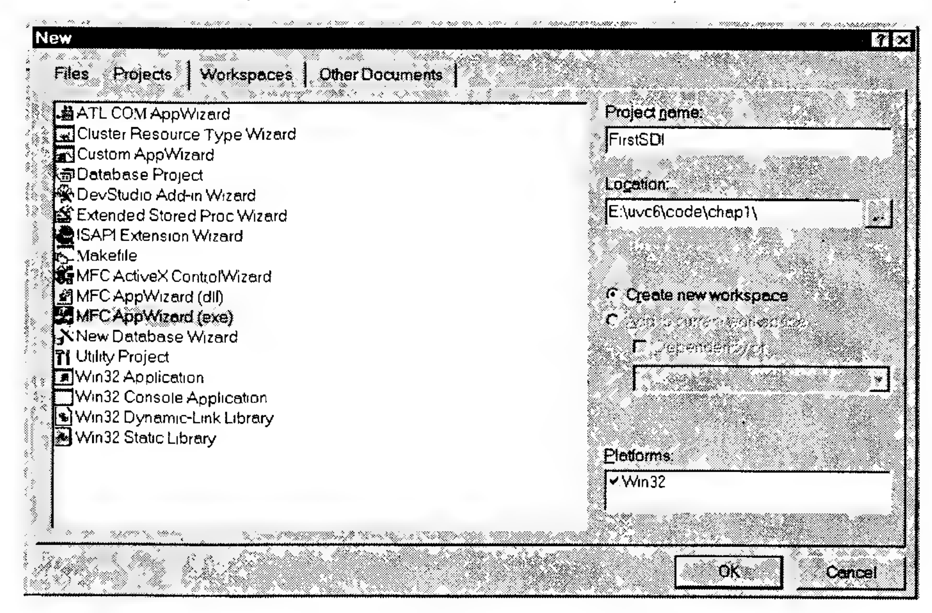


Рис. 1.1. На вкладке Projects *в окне* New *можно заказать тип приложения, которое вы хотите создать*

Из списка в левой части окна выберите MFC AppWizard (.exe), укажите имя проекта в поле Project name и щелкните на OK. Дальнейшие действия AppWizard пронумерованы как отдельные этапы (step), причем номер текущего этапа будет у вас всегда перед глазами в строке заголовка окна MFC AppWizard. На каждом этапе от пользователя требуется выбрать функцию создаваемого приложения, а затем щелкнуть на кнопке Next. На любой стадии можно вернуться к предыдущему этапу, щелкнув на кнопке Back. Если же щелкнуть на кнопке Can-

[2] По аналогии с архисовременной технологией в кулинарии такие заготовки программного кода называются *boilerplates* — "растворимый полуфабрикат" наподобие известных многим холостякам растворимых бульонов и молока. — *Прим. ред.*

cel, процесс создания приложения вообще будет прерван, а все предыдущие настройки отменены. Оперативная справка по текущему этапу вызывается на экран с помощью кнопки Help, а кнопка Finish позволяет завершить сеанс настройки, пропустив последующие этапы. К последнему средству мы рекомендуем прибегать только на заключительном этапе. Каждому этапу настройки AppWizard посвящен отдельный раздел этой главы.

На заметку *MFC-приложения* используют библиотеку классов Microsoft Foundation Classes (MFC). Описания и ссылки на отдельные компоненты этой библиотеки будут встречаться практически на каждой странице этой книги.

Выбор количества окон, которые будут поддерживаться приложением

Первое, что должен определить программист, приступая к работе в AppWizard, — сколько окон будет поддерживать будущее приложение, т.е. будет ли оно MDI-приложением, SDI-приложением или простым диалоговым приложением. Для каждого из этих типов приложений AppWizard по-разному создает программный код и классы. Окно MFC AppWizard при этом будет выглядеть так, как показано на рис. 1.2.

Подробности о каждом из этих типов приложений приведены ниже.

- SDI-приложение *(SDI — Single Document Interface*; интерфейс с единственным документом) позволяет в каждый момент времени иметь открытым только один документ. Примером может служить известный каждому редактор Notepad. Если вы выберете в таком приложении File⇨Open, то открытый в текущий момент файл будет закрыт прежде, чем откроется новый. Создание SDI-приложения настраивается в окне MFC AppWizard переключателем Single document.

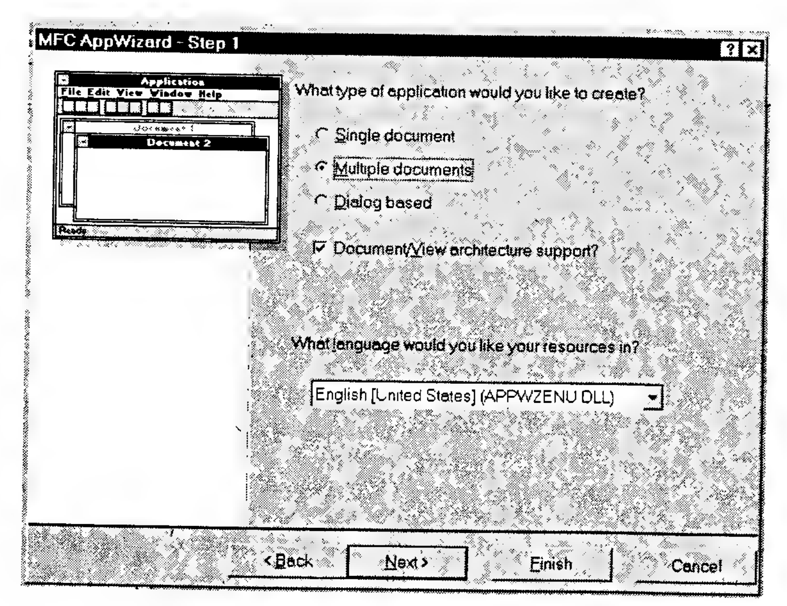


Рис. 1.2. Первый этап создания типового приложения с помощью AppWizard: выбор варианта интерфейса пользователя

- MDI-приложение (*MDI — Multiple Document Interface*; дословно —"многодокументный интерфейс") может одновременно держать открытыми несколько документов, каждый из которых представлен отдельным файлом. За примерами ходить далеко не надо — это и Excel, и Word, и другие хорошо знакомые многим аналогичные приложения. Такие приложения обязательно имеют в главном меню пункт Window, а в меню File — пункт Close. Одна из особенностей MFC состоит в том, что если необходимо создать приложение с несколькими представлениями (view) одного и того же документа, то его следует при настройке `AppWizard` отнести к типу MDI-приложений. Создание MDI-приложения настраивается в окне MFC AppWizard переключателем Multiple documents.
- Простое диалоговое приложение вообще не открывает документов. Примером может служить представленное на рис. 1.3 приложение `Character Map` и много других простых приложений, которые входят в базовый комплект Windows. Такие приложения не имеют меню. (Приложение `Character Map`, скорее всего, находится в папке `Accessories` (Стандартные), до которой можно добраться, щелкнув на кнопке Start (Пуск). Возможно, вам понадобится установить его на свой компьютер. Тогда воспользуйтесь функцией Add/Remove programs программы `Control Panel`.) Создание приложения этого типа настраивается в окне MFC AppWizard переключателем Dialog based.

Рис. 1.3. Приложение `Character Map` *является типичным примером простого диалогового приложения*

В левой части окна MFC AppWizard после выбора переключателя типа приложения появится соответствующий образец вашего будущего детища.

На заметку Простое диалоговое приложение достаточно сильно отличается от SDI-приложения, не говоря уже о MDI-приложении. Дальнейшие этапы создания приложения этого типа подробно рассматриваются ниже, в разделе *Создание простого диалогового приложения*.

Ниже этой группы переключателей в диалоговом окне находится флажок Document/View architecture support (Поддержка архитектуры документ/представление). Особенности архитектуры документ/представление будут подробно разъяснены в главе 4, которая специально посвящена этому вопросу. Пользователи, которые имеют большой опыт разработки приложений в среде Visual C++, могут отключить поддержку этой архитектуры мастером, но для большинства разработчиков она будет отнюдь не лишней. Поэтому в дальнейшем, если не будет оговорено особо, будем считать, что флажок Document/View architecture support установлен.

Еще ниже в окне MFC AppWizard находится раскрывающийся список для выбора языка, который вы собираетесь использовать при написании текста программы. Если системный язык операционной среды, не заданный по умолчанию `English (United States)`, — американский вариант английского, не забудьте сделать такой же выбор и в списке. Иначе можете в дальнейшем столкнуться с совершенно неожиданными эффектами в работе с `ClassWizard`.

(Конечно, если вы создаете приложения для заказчика, который пользуется американским английским, у вас не остается иного выбора, как изменить системный язык с помощью программы Control panel.) Для перехода к следующему диалоговому окну мастера щелкните на кнопке Next.

Базы данных

Второй этап создания приложения Windows с помощью AppWizard — выбор уровня поддержки операций с базами данных. Окно MFC AppWizard при этом будет выглядеть так, как показано на рис. 1.4.

Здесь вам на выбор предлагается четыре варианта уровня поддержки.

- Если работа с базами данных в приложении не планируется, выберите переключатель None (Никакой).
- Если предполагается иметь доступ к базам данных, но для этого не будут использованы классы просмотра, производные от CFormView, или нет необходимости в меню Record (Запись), выбирайте переключатель Header files only (Только файлы заголовков).
- Если вы планируете разрабатывать классы просмотра базы данных в приложении как производные от CFormView и иметь меню Record, но не нуждаетесь в средствах сохранения-восстановления документов, выбирайте переключатель Database view without file support (Просмотр базы данных без поддержки операций с файлами). Можно будет обновлять записи в базе данных с помощью CRecordset — класса MFC, который подробно обсуждается в главе 22, *Доступ к базам данных*.
- Если, помимо всего, что задано в предыдущем случае, вы планируете и сохранение-восстановление документов на диске (возможно, это будет одна из опций пользователя), выберите Database view with file support (Просмотр базы данных и поддержка операций с файлами).

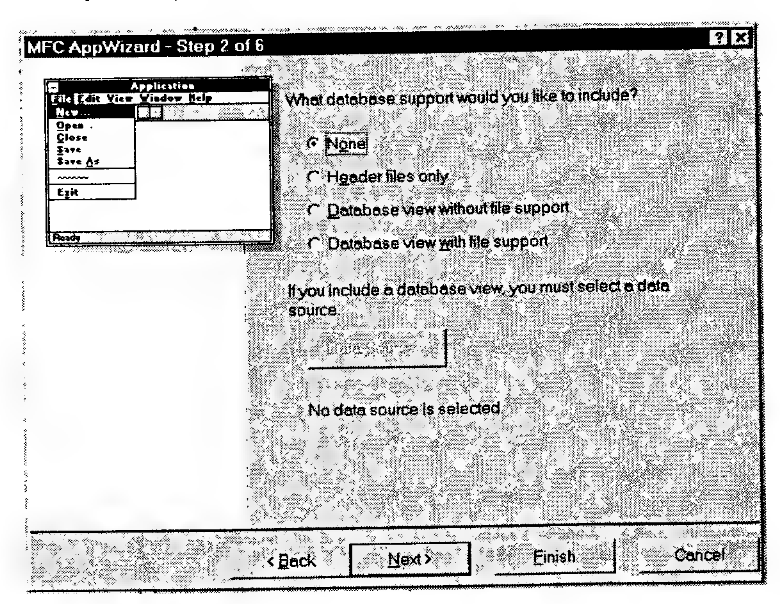


Рис. 1.4. Второй этап создания типового приложения с помощью AppWizard: выбор варианта поддержки операций с базами данных

В главе 22, *Доступ к базам данных*, более подробно описываются последствия каждого варианта выбора и демонстрируется методика программирования баз данных с использованием MFC. Если вы выбрали один из вариантов с использованием базы данных, в этом же окне задайте тип базы. Для этого нужно щелкнуть на кнопке Data Source (Источник данных).

Картинка в левой части окна MFC AppWizard меняется после задания любого из предложенных вариантов обращения к базе данных, демонстрируя последствия сделанного выбора. Для перехода к следующему диалоговому окну мастера щелкните на кнопке Next.

Поддержка составных документов

Третий этап создания выполняемого приложения Windows с помощью AppWizard — выбор уровня поддержки операций с составными документами. Окно MFC AppWizard при этом будет выглядеть так, как показано на рис. 1.5. Технология OLE (Object Linking and Embedding — связывание и внедрение объектов) в настоящее время получила новое название — *технология ActiveX*. Оно подчеркивает направление развития этой технологии, скрытое от большинства программистов библиотекой классов MFC. И та, и другая технологии представляют собой некоторое подмножество *технологии составных документов (compound document technology)*. Подробно технология ActiveX рассматривается в главе 13, *Концепции технологии ActiveX*.

На выбор предлагается пять вариантов поддержки.

- Если не планируется создание ActiveX-приложения, выберите переключатель None (Никакой).
- Если вы хотите, чтобы в приложении использовались связанные или внедренные объекты ActiveX, такие как документы Word или рабочие листы Excel, выберите переключатель Container (Контейнер). Как строить ActiveX-контейнер, вы узнаете в главе 14, *Создание приложения-контейнера ActiveX*.

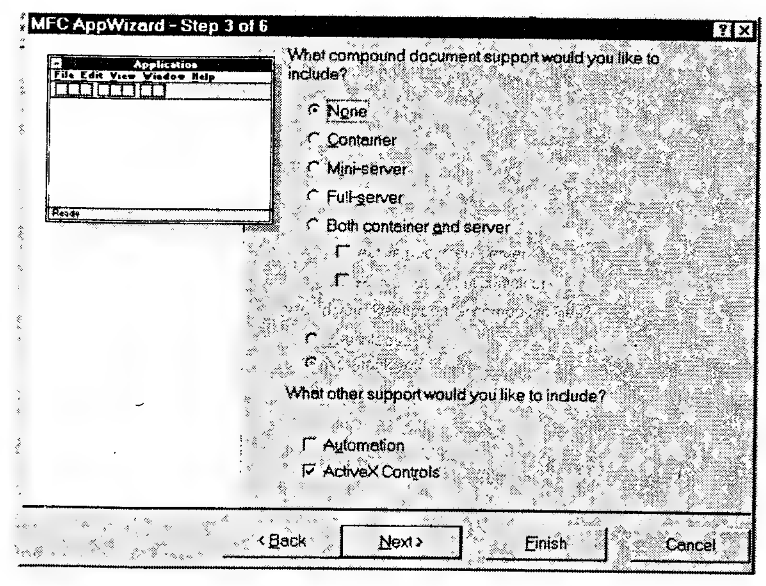


Рис. 1.5. Третий этап создания типового приложения с помощью AppWizard: выбор варианта поддержки технологии составных документов

- Если планируется создание приложения, документы которого могли бы быть внедрены в другое приложение, но при этом само приложение не будет использоваться автономно, выберите переключатель Mini-server (Мини-сервер).
- Если ваше будущее приложение будет служить не только сервером для других приложений, но и сможет работать автономно, выберите переключатель Full-server (Полный сервер). Как строить полный сервер ActiveX, вы узнаете в главе 15, *Создание приложения-сервера ActiveX*.
- И, наконец, если создаваемое приложение должно обладать способностью включать документы других приложений и само обслуживать их своими объектами, выбирайте переключатель Both container and server (И контейнер, и сервер).

Если уж вы выбрали какой-либо из вариантов поддержки составных документов, то придется поддерживать и *составные файлы (compound files)*. Составные файлы содержат один или более объектов ActiveX и сохраняются на диске в особом формате, так что один из объектов может быть заменен без переписи всего файла. Таким образом удается сберечь довольно много времени. В середине окна MFC AppWizard Step 3 имеется группа Would you like to support compound files? (Не угодно ли вам заказать поддержку составных файлов?) из двух переключателей. Если вы уже приняли соответствующее решение, то смело щелкайте либо на переключателе Yes, please (Да, будьте любезны), либо на No, thank you (Спасибо, я, пожалуй, воздержусь).

Если вы хотите, чтобы создаваемое приложение могло передавать управление другому приложению через механизм автоматизации ActiveX, установите флажок Automation (Автоматизация). (Механизм автоматизации ActiveX подробно рассматривается в главе 16, *Создание сервера автоматизации*.) Если планируется использовать в приложении элементы управления ActiveX, установите флажок ActiveX Controls (Элементы управления ActiveX). А теперь щелкните на кнопке Next для того, чтобы перейти к следующему этапу.

На заметку Если хотите, чтобы создаваемое приложение само было элементом управления ActiveX, то все описываемое в этой главе вас не касается, поскольку вам не нужно заказывать типовое приложение (Exe-файл). Технология создания с помощью мастера ControlWizard приложений, которые являются элементами управления ActiveX, описана в главе 17, *Создание элемента управления ActiveX*.

Внешний вид приложения и другие опции

Четвертый этап создания выполняемого приложения Windows с помощью AppWizard — выбор опций, определяющих внешний вид элементов пользовательского интерфейса. Окно MFC AppWizard при этом будет выглядеть так, как показано на рис. 1.6.

Диалоговое окно MFC AppWizard — Step 4 of 6 содержит много переключателей-флажков, соответствующих предлагаемым опциям оформления.

- *Docking toolbar* (Фиксируемая панель инструментов). AppWizard установит в приложении панель инструментов, которая может быть пристыкована (зафиксирована) к одной из границ окна. Затем можно будет удалить ненужные пиктограммы из панели или добавить новые, связанные с теми пунктами меню, которые вы посчитаете нужными включить в свое приложение. Вся технология разработки меню и панелей управления в приложении описана в главе 9, *Панели инструментов и строка состояния*.
- *Initial status bar* (Панель состояния). AppWizard создаст в приложении панель состояния, в которой можно будет выводить подсказки соответственно выбранным пунктам меню и другие сообщения. Позднее можно будет написать специальные программки для вывода различных индикаторов на эту панель, о чем будет подробно рассказано в той же главе 9.

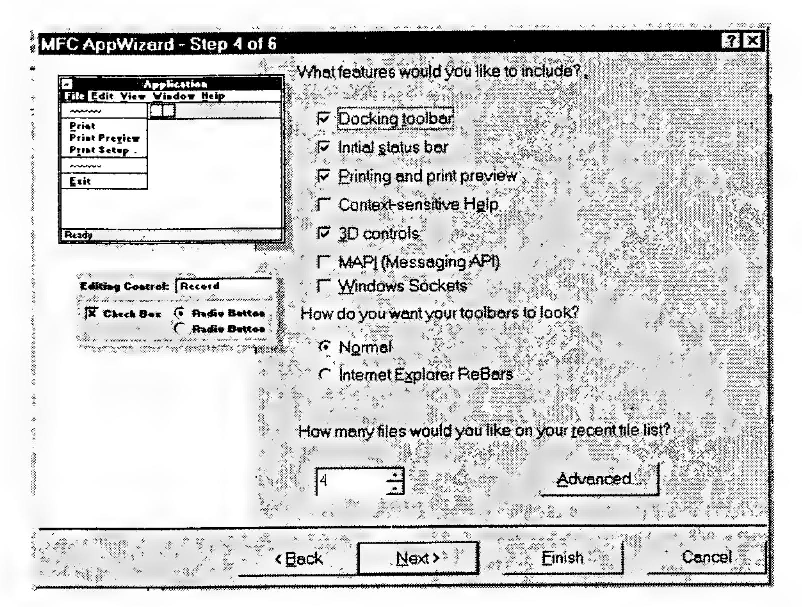


Рис. 1.6. Четвертый этап создания типового приложения с помощью AppWizard. установка некоторых опций экстерьера пользовательского интерфейса

- *Printing and print preview* (Печать и предварительный просмотр распечатки). Приложение при выборе этой опции будет иметь пункты Print и Print preview в меню File, и AppWizard включит в приложение большую часть программного кода, связанного с выполнением этих операций. Об остальном вы узнаете из главы 6, *Распечатка и предварительный просмотр.*
- *Context sensitive Help* (Контекстная справка). Меню Help в приложении будет иметь опции Index и Using Help, а значительная часть программного кода, необходимого для организации контекстной справки в приложении, будет включена в него мастером AppWizard. С методикой организации контекстной справки в приложении вы сможете познакомиться подробно в главе 11, *Справка в приложении.*
- *3D controls* (Объемный дизайн элементов управления). При установке этой опции дизайн приложения будет полностью соответствовать стилю, принятому в фирменных приложениях Windows 95. Если вы откажетесь от этой опции, то фон диалоговых окон будет белым, а такие элементы, как текстовые поля, переключатели и вкладки, не будут отбрасывать тени.
- MAPI (Messaging API) (MAPI — почтовый интерфейс). При установке этой опции приложение сможет обмениваться сообщениями по электронной почте и отправлять факсы. Обсуждение подробностей реализации MAPI мы откладываем до главы 18, *Windows Sockets, MAPI и Internet.*
- *Windows Sockets.* Если эта опция будет установлена, приложение сможет иметь непосредственный доступ к Internet через такие протоколы, как FTP и HTTP (протокол World Wide Web). Концепция Windows Sockets обсуждается в главе 18. Можно создать Internet-программу и без поддержки Windows Sockets, если использовать новые классы WinInet, о чем подробно будет рассказано в главе 19, *Использование классов WinInet при программировании для Internet.*

Мастеру AppWizard можно заказать создание панелей инструментов в традиционном стиле, как в Word или в самом продукте Visual C++, или в новом стиле оформления, принятом в Internet Explorer. Более подробно об этом будет сказано в главе 9.

Можно также установить длину списка самых "свежих" файлов в поле меню File создаваемого приложения. Для этого служит раскрывающийся список How many files would you like on your recent file list?. По умолчанию этот параметр имеет значение 4 и менять его не рекомендуется без очень весомых причин.

После щелчка на кнопке Advanced (Дополнительно) в нижней части диалогового окна MFC AppWizard Step 4 на экран будет выведено новое диалоговое окно Advanced Options (Дополнительные опции), которое имеет две вкладки. На рис. 1.7 показана одна из них — Document Template Strings (Строковые шаблоны документов). Дело в том, что AppWizard формирует многочисленные запросы и идентификаторы, принимая в качестве главного переменного элемента имя приложения и иногда ему необходимы аббревиатуры этого имени. До тех пор, пока вы не усвоите, как AppWizard строит такие имена, вам придется справляться о них в этом окне. Здесь же их можно при желании откорректировать, а также уточнить надпись, которая будет выведена в строке заголовка главного окна создаваемого приложения. Расширения имени файла, если вы установите его в поле File extension, будут автоматически добавляться к именам всех файлов, которые записываются на диск приложением. Аналогично по команде File⇒Open в соответствующем диалоговом окне будут выведены по умолчанию только файлы с заданным расширением.

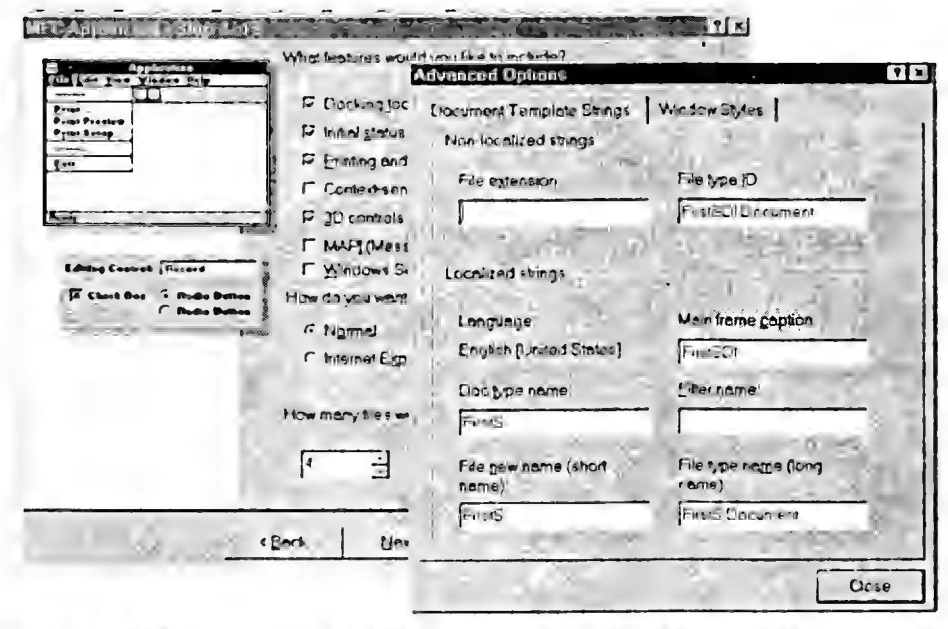


Рис. 1.7. Вкладка Document Template Strings диалогового окна Advanced Options позволяет уточнить некоторые сокращения имен в приложении

На рис. 1.8 показана другая вкладка — Window Styles (Стили оформления окон). В этом окне можно кардинально изменить дизайн окон приложения. Первый флажок — Use Split Window (Использование разделения окна). При его установке в приложение включается весь программный код, необходимый для организации разделения окна приложения таким же образом, как это сделано, например, в редакторе программного кода из комплекта средств Visual Studio. Остальные элементы диалогового окна устанавливают параметры, определяю-

щие оформление фрейма (рамки) *главного окна* приложения, а для MDI-приложений — фреймов *дочерних окон* (child frames). Фрейм является весьма важным элементом окна; системное меню, строка заголовка, кнопки минимизации и максимизации, собственно границы — все это *свойства* фрейма как объекта. Фрейм главного окна содержит все SDI-приложение. MDI-приложение имеет несколько *дочерних* окон (по одному на каждый документ), которые размещаются в пределах главного, *родительского*, окна.

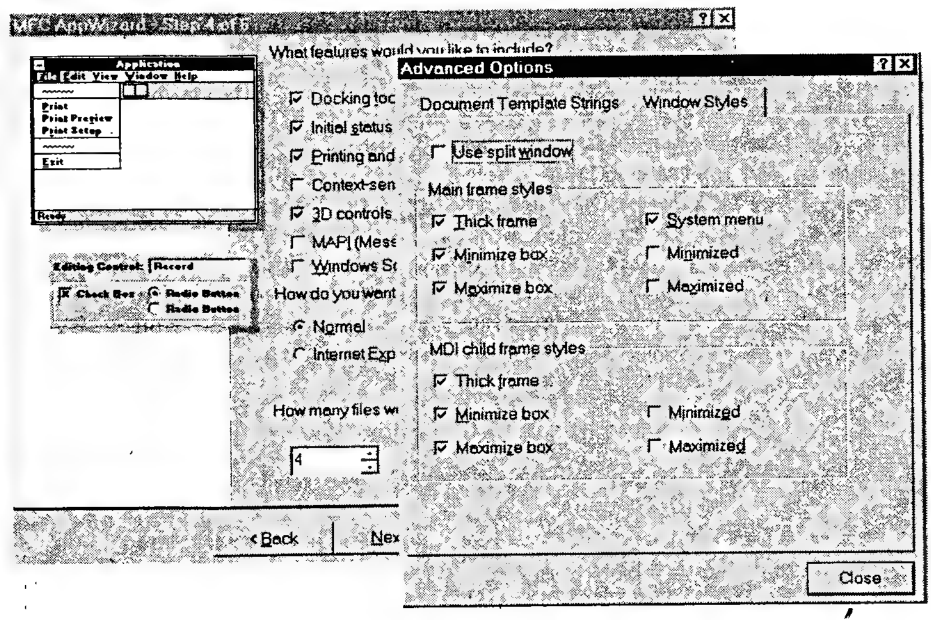


Рис. 1.8. Вкладка Window Styles *диалогового окна* Advanced Options *позволяет настроить экстерьер окон приложения*

Ниже перечислены свойства фрейма, которые можно настраивать во вкладке, о которой идет речь.

- *Thick frame* (утолщенная рамка). Кромки окна утолщены, и можно будет изменять размеры окна стандартным для Windows способом.
- *Minimize box* (кнопка минимизации). Окно имеет кнопку минимизации в правой части строки заголовка.
- *Maximize box* (кнопка максимизации). Окно имеет кнопку максимизации в правой части строки заголовка.
- *System menu* (системное меню). В левом верхнем углу окна будет установлена пиктограмма вызова системного меню.
- *Minimized*. При запуске приложения окно сворачивается в пиктограмму. Для SDI-приложений выбор этой опции не будет иметь никаких последствий при выполнении приложения в среде Windows 95.
- *Maximized*. При запуске приложения окно разворачивается на весь экран. Для SDI-приложений выбор этой опции не будет иметь никаких последствий при выполнении приложения в среде Windows 95.

После завершения всех манипуляций щелкните на Close для возврата в окно MFC AppWizard — Step 4 of 6, а затем на Next — и перейдете к следующему этапу.

Другие опции

Пятый этап создания выполняемого приложения Windows с помощью AppWizard — выбор опций, которые нельзя было отнести по назначению ни к одному из предыдущих этапов. Окно MFC AppWizard — Step 5 of 6 при этом будет выглядеть так, как показано на рис. 1.9. Будете ли вы включать в формируемый текст программ приложения комментарии? Редко кто отказывается от этого, тем более что задать такой режим не составляет никакого труда. Нужно просто выбрать один из переключателей группы Would you like to generate source file comments? (Не будет ли вам угодно включить в текст программы комментарии?).

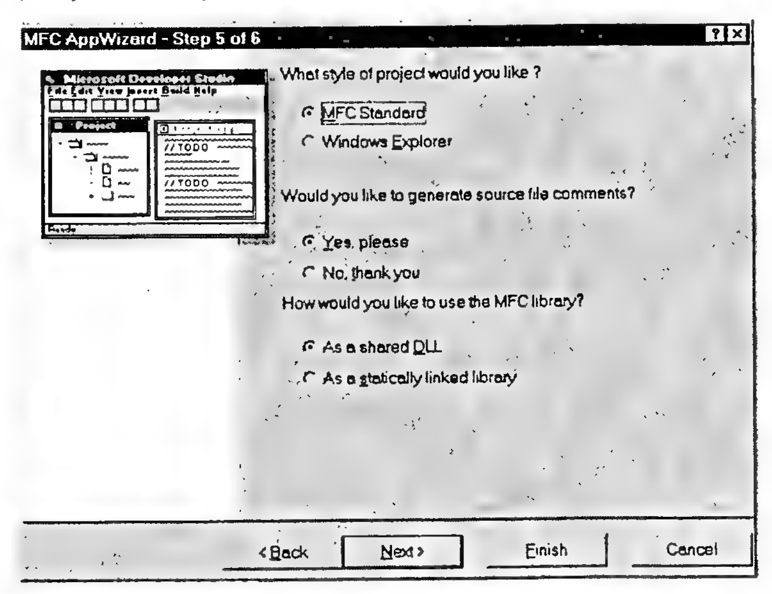


Рис. 1.9. Пятый этап создания типового приложения с помощью AppWizard: установка опций, определяющих оформление формируемого текста программы и метод связывания программы с модулями библиотеки MFC

Ответ на второй вопрос в этом окне не так очевиден. Желаете ли вы, чтобы библиотека MFC была *разделяемой динамически связываемой библиотекой* (shared DLL) или *статически прикомпонованной* (statically linked library)? Динамически связываемая библиотека (DLL — Dynamic-Link Library) представляет собой множество функций, используемых самыми разными приложениями. Использование DLL сокращает объем программы, но несколько усложняет установку продукта. Если вы просто перенесете на другой компьютер выполняемый файл программы, то, скорее всего, приложение работать не будет, поскольку оно нуждается еще и в соответствующих DLL-файлах. Если же модули библиотеки прикомпонованы статически к выполняемому файлу, то приложение легко перемещается с одного компьютера на другой, поскольку весь выполняемый код сосредоточен в одном файле.

Если вы ориентируетесь на заказчиков, которые сами являются разработчиками и имеют на своем компьютере по крайней мере одно приложение, использующее модули DLL из библиотеки MFC, или ваши заказчики не против установки DLL-файлов на своем компьютере, смело выбирайте опцию As a shared DLL. Более короткий EXE-файл — это всегда хорошо. Если же ваши потенциальные пользователи не столь искушены в компьютерных премудростях, лучше используйте опцию As a statically linked library. В результате получится более длинный EXE-

файл, но вы будете избавлены от многих забот при сопровождении продукта у не слишком-то опытного пользователя. И последнее замечание. Если вы разработаете достаточно хорошую программу установки продукта на ПК пользователя, то почувствуете, что многие сложности с использованием динамически связываемых библиотек на самом деле не так уж и страшны.

Имена файлов и классов

И наконец, последний этап создания выполняемого приложения Windows с помощью AppW.zard — подтверждение имен классов и имен файлов, которые создает для вас AppWizard, как это показано на рис. 1.10. AppWizard использует имя проекта (в данном случае — FirstSDI) для формирования имен классов и имен файлов. Нет никакой нужды их изменять. Если в приложении используются классы представления, можно изменить имя класса, наследниками которого являются вновь создаваемые классы. По умолчанию базовым является CView, но многие разработчики предпочитают CScrollView или CEditView. В главе 4, *Документы и представления*, детально рассматриваются классы представления документов. После завершения работы в окне MFC AppWizard — Step 6 of 6 щелкните на Finish.

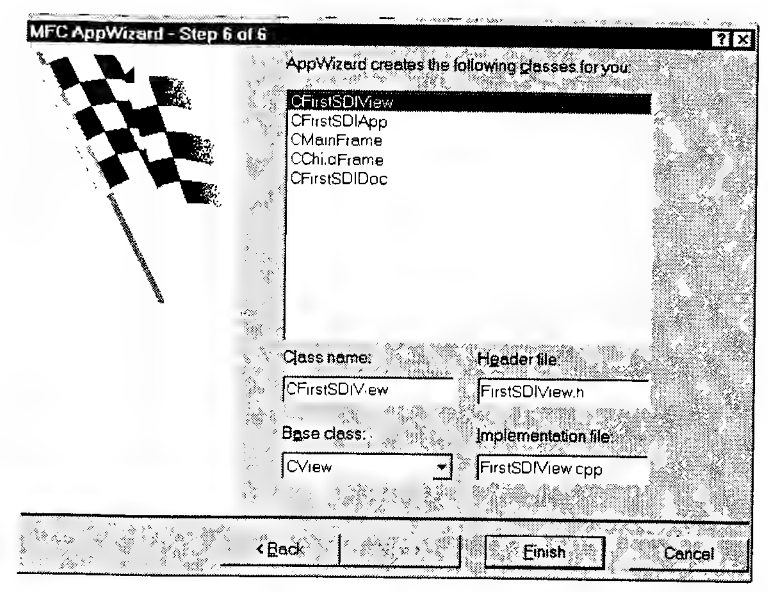


Рис. 1.10. Последний этап создания типового приложения с помощью AppWizard. подтверждение имен классов и файлов



Вы можете освежить свои знания об объектах, классах и наследовании, заглянув в приложение А, *Обзор языка C++ и основные концепции объектно-ориентированного программирования*.

Создание приложения

После того как вы щелкнете на Finish, AppWizard покажет вам в окне New Project Information (Информация о новом проекте), что же он собирается создавать (рис. 1.11). Если что-либо здесь вас не устраивает, щелкните на Cancel и затем последовательно двигайтесь в обратном порядке по окнам этапов настройки, щелкая на Back (Назад), пока не выйдете на то окно, в котором желательно изменить настройку. После уточнения настройки снова последо-

вательно "прощелкайте" этапы с помощью кнопок Next и Finish, еще раз взгляните на окно New Project Information и уж после этого смело щелкайте на OK. AppWizard приступит к собственно созданию приложения. Это займет несколько минут. За это время, пока вы будете пить кофе, AppWizard создаст сотни строк программного кода, меню, диалоговых окон, текстов справок, растровых картинок, которые будут записаны в более чем 20 различных файлов. Пусть себе работает...

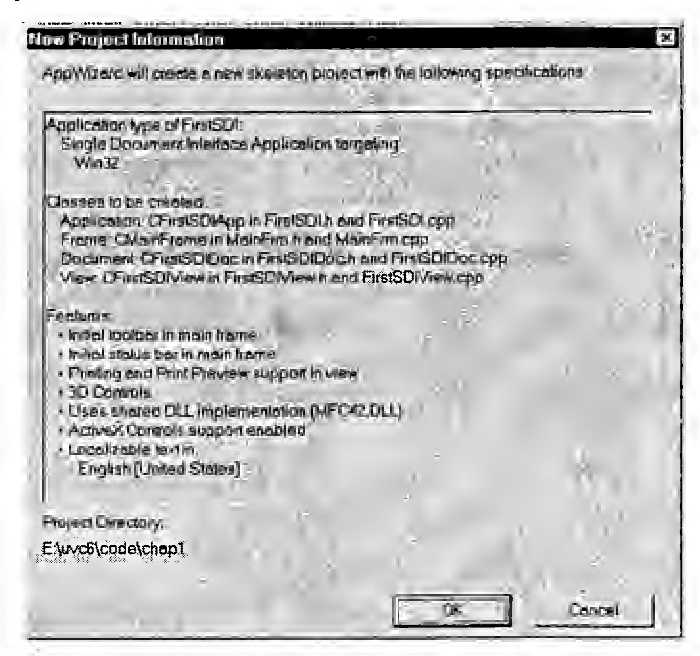


Рис. 1.11. Когда AppWizard готов приступить к собственно созданию приложения, он дает вам еще один шанс проверить все настройки

Поработайте самостоятельно

Если вы еще не запустили Visual Studio, самое время это сделать. Если прежде вы никогда не пользовались этим продуктом, то его интерфейс может показаться, на первый взгляд, слегка пугающим. Полное описание всех областей, панелей, меню и ускорителей приведено в приложении В, *Интерфейс Visual Studio*.

Первым делом вызовите AppWizard. Для этого нужно выбрать File⇨New и щелкнуть на вкладке Projects. На этой вкладке введите имя папки, в которой вы собираетесь хранить все файлы создаваемого приложения, — AppWizard открывает новую папку для каждого нового приложения. В качестве имени проекта введите FirstSDI, а затем пройдите все 6 этапов настройки параметров нового приложения с помощью AppWizard. На первом этапе выберите тип приложения SDI, а на всех последующих не меняйте установки, сделанные по умолчанию, и просто щелкайте на Next. Когда AppWizard закончит работу над проектом, выберите Build⇨Build из меню Visual Studio. По этой команде приложение будет откомпилировано и скомпоновано.

Когда этот процесс завершится, выберите из меню Visual Studio Build⇨Execute. Перед вами появится реальное, работоспособное приложение Windows, показанное на рис. 1.12. Сначала "поиграйте" с ним, попробуйте изменить размеры и положение окна, свернуть его в пиктограмму или развернуть на весь экран.

Поработайте с меню File — выберите File⇨Open, и перед вами распахнется до боли знакомое диалоговое окно Windows File Open. Можете смело выбирать любой файл, ваше приложение с ним ничего не сделает. Установите указатель мыши на одной из пиктограмм панели инструментов и задержите его там на некоторое время. Появится маленькое *контекстное окно указателя* (tool tip), текст, который напомнит о назначении этой пиктограммы. Для проверки, действительно ли пиктограммы панели инструментов дублируют те или иные команды меню, щелкните на пиктограмме Open и убедитесь, что все дальнейшее точно повторяет возникшую чуть раньше ситуацию, когда вы выбрали из меню File⇨Open. Теперь обратитесь к меню View и щелкните на уже ранее отмеченном пункте Toolbar (Панель инструментов). Панель исчезнет с экрана. Выберите View⇨Toolbar — и панель инструментов вновь появится на экране. То же самое сделайте с панелью состояния. Выберите Help⇨About; возможно, вы этого не ожидали, но созданное практически без вашего участия приложение имеет даже окно сообщения об авторских правах (рис. 1.13). Когда вдоволь натешитесь, выберите File⇨Exit и закройте приложение.

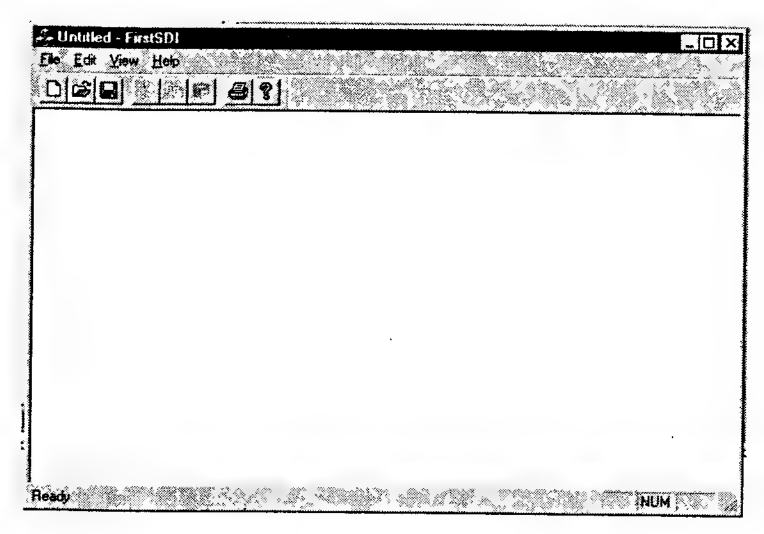


Рис. 1.12. Внешне ваше первое приложение ничем не отличается от любого "вполне оперившегося" приложения Windows

Теперь проведем аналогичный эксперимент с приложением типа MDI, которое назовем FirstMDI. Отличие в процессе создания приложения будет только на этапе 0, на котором задается имя проекта, и на этапе 1, на котором придется выбрать переключатель Multiple documents. На остальных этапах можно вполне согласиться с настройками, предлагаемыми AppWizard по умолчанию. После завершения работы AppWizard повторите уже знакомые операции с Visual Studio и запустите приложение. Вы увидите на экране нечто, весьма близкое к изображенному на рис. 1.14, — MDI-приложение с одним открытым документом. Попробуйте провести те же эксперименты, что и с предыдущим приложением FirstSDI.

Выберите File⇨New — и появится второе окно FirstM2. Поэкспериментируйте теперь с разными окнами приложения — сверните, разверните на весь экран и приведите их к первоначальному состоянию. Между окнами можно переключаться, обращаясь к меню Window. AppWizard дарит вам все эти функциональные возможности, не требуя взамен ни единой строки текста программы.

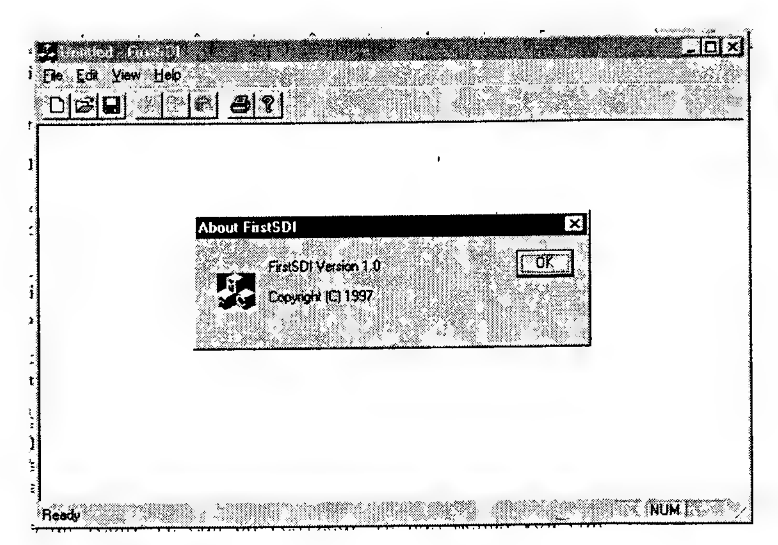


Рис. 1.13. В этом приложении есть даже окно сообщения об авторских правах

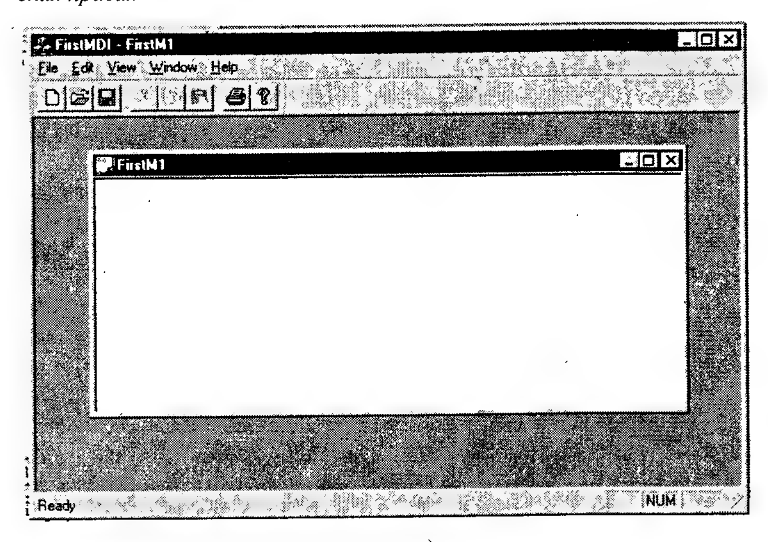


Рис. 1.14. MDI-приложение может одновременно вывести на экран несколько документов

Создание простого диалогового приложения

Приложение этого типа не имеет никакого меню, кроме системного, и не может открывать или сохранять файлы. Это оптимальный вариант для относительно простых утилит наподобие Character Map, которая входит в состав базового комплекта Windows. Процесс построения такого приложения с помощью AppWizard несколько отличается от рассмотренного ранее, в первую очередь, по той простой причине, что в этом случае нет смысла задавать "заумные" вопросы о поддержке баз данных или составных документов. Процесс построения начинается так же, как и для SDI- и MDI-приложений. Запустите AppWizard, но на этапе 1 выберите переключатель Dialog based, как это показано на рис. 1.15.

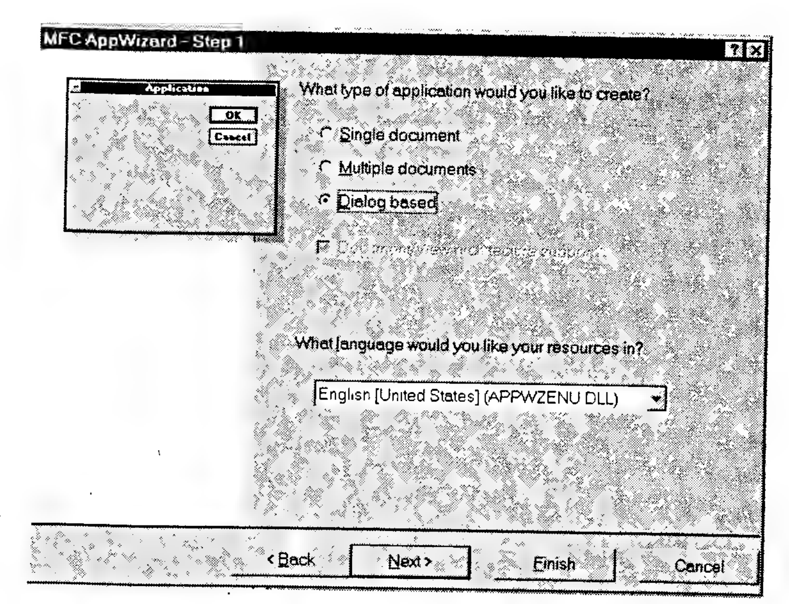


Рис. 1.15. Для создания простого диалогового приложения выберите соответствующую настройку в окне MFC AppWizard — Step 1

После этого щелкните на кнопке Next и перейдите к этапу 2, как это показано на рис. 1.16.

Если вы решили включить в системное меню пункт About, установите флажок About box. Для того чтобы AppWizard подготовил все необходимое для включения справки в приложение, нужно установить флажок Context-sensitive Help (Контекстная справка). Третий флажок в этом окне — 3D controls (Объемный дизайн элементов управления) — рекомендуется устанавливать для большинства приложений, работа которых планируется в операционных средах Windows 95 и Windows NT. Если вы хотите, чтобы создаваемое приложение могло передавать управление другому приложению через механизм автоматизации ActiveX, который подробно рассматривается в главе 16, установите флажок Automation. Если планируется использовать в приложении элементы управления ActiveX, установите флажок ActiveX Controls (элементы управления ActiveX). Если планируется, что приложение будет иметь доступ к Internet через Windows Sockets, установите флажок Windows Sockets. (Простое диалоговое приложение не может использовать почтовые средства MAPI по той простой причине, что оно не оперирует документами.) Щелкните на кнопке Next для того, чтобы перейти к третьему этапу, окно которого представлено на рис. 1.17.

При создании простых диалоговых приложений, как и их более сложных собратьев, о которых шла речь выше, AppWizard может включить в текст программ комментарии. Что касается решения о выборе между статической компоновкой и разделяемыми DLL-модулями библиотечных функций MFC, то для данного варианта приложения доводы в пользу каждой из альтернатив сохраняются теми же, что и для SDI- и MDI-приложений. Если ваш потенциальный пользователь уже имеет DLL-модули на своем компьютере (он сам является разработчиком программного обеспечения или пользуется другим программным продуктом, обращающимся к DLL-модулям) или он не против того, чтобы ваш продукт был в этом смысле первопроходцем на его компьютере, выбирайте в группе How would you like to use MFC library? переключатель As a shared DLL с тем, чтобы EXE-файл получился покороче, а компоновка выполнялась быстрее. В противном случае выбирайте в этой группе переключатель As a statically linked library. Щелкните на кнопке Next для того, чтобы перейти к последнему этапу настройки AppWizard, окно которого представлено на рис. 1.18.

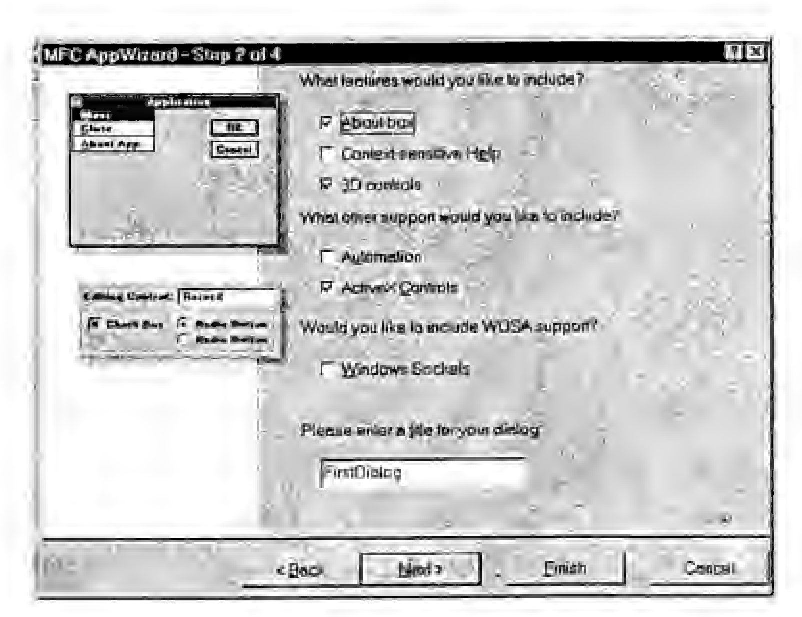


Рис. 1.16. Этап 2 настройки AppWizard для создания простого диалогового приложения включает выбор параметров для системы справки, элементов ActiveX, автоматных серверов и интерфейса Sockets

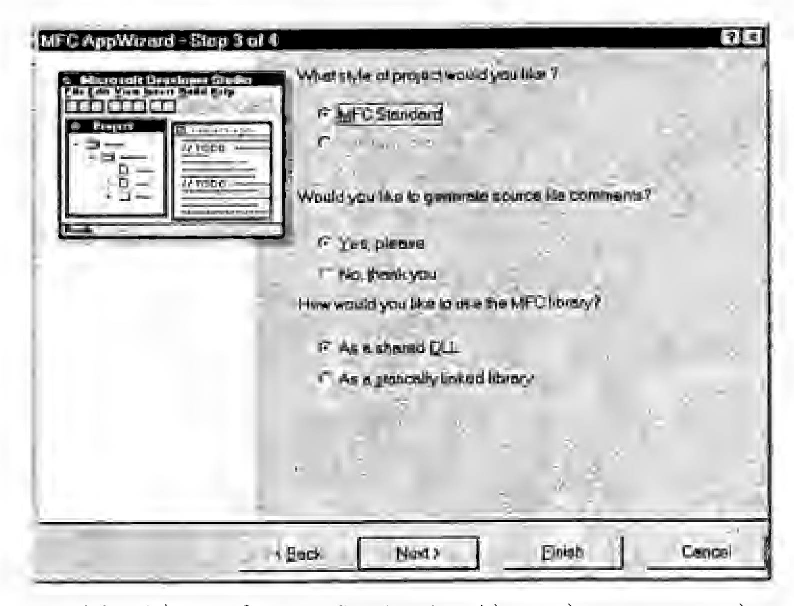


Рис. 1.17. На этапе 3 настройки AppWizard для создания простого диалогового приложения вы будете иметь дело с комментариями в тексте программы и способом компоновки библиотеки MFC

На этом этапе можно уточнить имена, выбранные AppWizard для файлов и классов. Редко кому удается придумать более удачные, поскольку тех, кто будет работать с вашей программой, может несколько смутить отсутствие видимой связи между именем какого-либо файла и именем соответствующего класса, или наоборот. Если вам покажется, что имя проекта было выбрано не очень удачно, вернитесь обратно в окно New Project Workspace (Рабочее пространство нового проекта) (для этого существует кнопка Back), измените имя проекта, щелкните на Create, а затем снова пощелкайте на Next, чтобы вернуться в окно последнего этапа творения. После завершения работы щелкните на Finish, и AppWizard представит вам итоговую спецификацию заказа всех классов и файлов будущего приложения, как это показано на рис. 1.19.

Если что-либо здесь вас не устраивает, щелкните на Cancel и затем щелкайте на Back, пока не выйдете на то окно, в котором желательно изменить настройку. После завершения настройки щелкните на OK и спокойно наблюдайте за процессом.

А теперь поработайте самостоятельно. Создайте прототип простого диалогового приложения, назовите его FirstDialog и не изменяйте настроек, которые AppWizard делает по умолчанию на каждом этапе для приложения такого типа. Когда закончите настройку мастера, выберите Build⇨Build из меню Visual Studio, и приложение будет откомпилировано и скомпоновано. Чтобы посмотреть, как оно будет работать, выберите Build⇨Execute. На рис. 1.20 показано, что вы увидите после этого на экране.

Если вы щелкнете на кнопке OK или Cancel, либо на кнопке, помеченной косым крестиком в правом углу строки заголовка (стандартная для Windows кнопка Close (Закрыть)), созданное вашими и AppWizard усилиями диалоговое окно исчезнет. Если щелкнуть на пиктограмме в левой части строки заголовка, откроется системное меню, в котором будут пункты Move (Передвинуть), Close и About (О программе). На рис. 1.21 показано окно About, которое настроено AppWizard по умолчанию и выведено на экран после обращения к команде About системного меню.

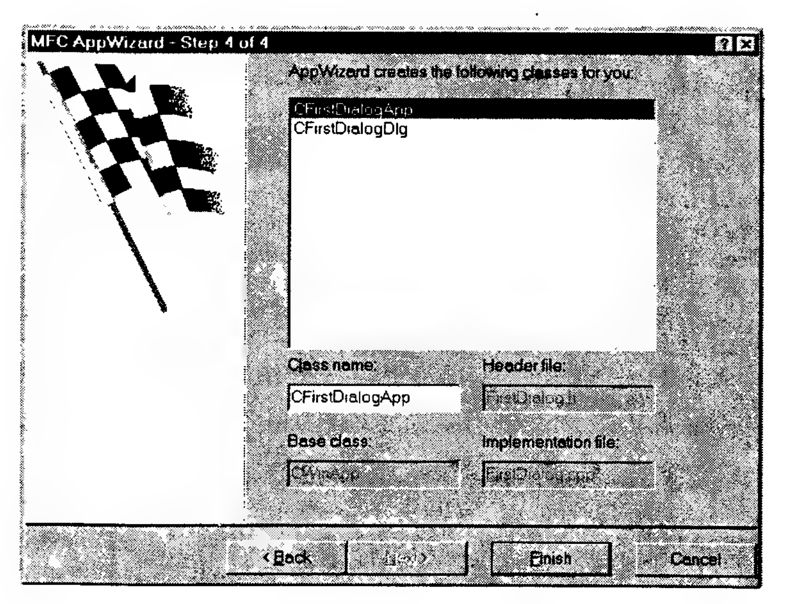


Рис. 1.18. Последний этап создания простого диалогового приложения с помощью AppWizard дает вам шанс уточнить имена классов и файлов

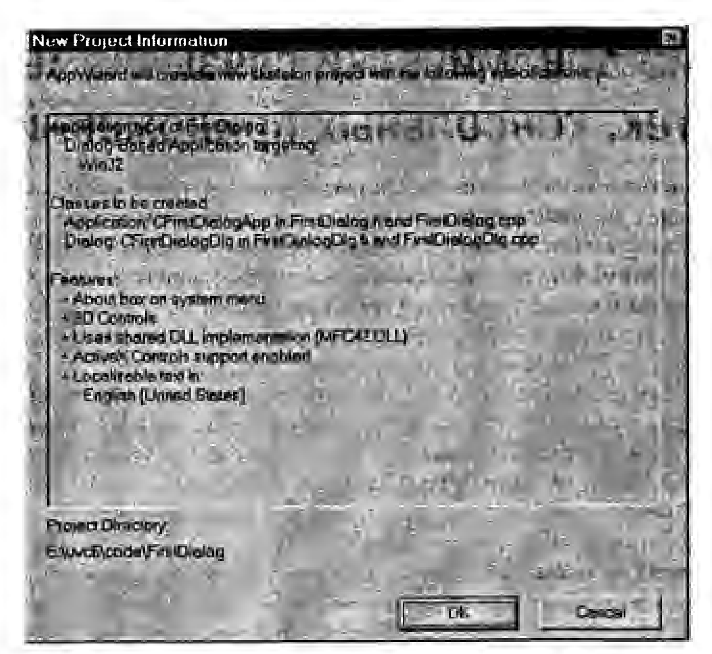


Рис. 1.19. *AppWizard предъявляет сводку всех выполненных настрое прежде, чем приступить к собственно созданию приложения*



Рис. 1.20. Прототип простого диалогового приложения напомнит вам о том, что еще предстоит сделать. Надпись в окне — СДЕЛАТЬ: Установите здесь элементы управления окна



Рис. 1.21. Окно сообщения об авторских правах для SDI-, MDI- и простого диалогового приложений

Создание динамически связываемых библиотек, консольных приложений и т.п.

Хотя большинство программистов используют AppWizard для создания выполняемых программ, он способен подготовить и другие проекты. Вам нужно выбрать File⇨New, а затем — вкладку Projects в окне New, как об этом уже было сказано в начале этой главы, но затем выбрать другой варнант Wizard (Мастера) из списка, показанного на рис. 1.1. Ниже прнводится перечень вариантов мастеров для других видов проектов.

- ATL COM AppWizard
- Custom AppWizard
- Database Project
- DevStudio Add-in Wizard
- Extended Stored Procedure AppWizard
- ISAPI Extension Wizard
- Makefile
- MFC ActiveX ControlWizard
- MFC AppWizard (DLL)
- Utility Project
- Win32 Application
- Win32 Console Application
- Win32 Dynamic-Link Library
- Win32 Static Library

В последующих разделах кратко поясняется назначение каждого из перечисленных видов проектов и особенности соответствующих типов мастеров.

Мастер ATL COM AppWizard

Аббревиатура ATL означает *Active Template Library* (библиотека активных шаблонов). Эта библиотека используется для подготовки программ обслуживания несложных элементов управления ActiveX. В основном к ее услугам прибегают разработчики, имеющие достаточно большой опыт создания средств управления MFC ActiveX. В главе 17 описаны основные концепции построения таких средств, которые иллюстрируются примерами использования соответствующих возможностей MFC. Прочитав и усвоив материал главы 21, *Библиотека Active Template Library*, вы научитесь пользоваться этой библиотекой.

Мастер Custom AppWizard

Возможно, вам повезло и вы работаете в большой компьютерной фирме, где занимаются разработкой множества различных программных продуктов. Хотя мастер AppWizard и позволяет сэкономить много времени, он не всемогущ, и, приступая к новому проекту, ваши коллеги могут потратить день или два (а это не так мало в нашем динамичном мире) на то, чтобы скопировать уже имеющиеся у вас наработки, которые можно будет в этом новом проекте использовать. После несложных рассуждений можно прийти к заключению, что есть смысл потратить немного времени и создать проект вида Custom AppWizard, т.е. специализированный мастер, который можно поместить в набор "растворимых блюд" наряду со стандартными компонентами MFC. После того как вы реализуете свой замысел, новое приложение появится

в списке в левой части вкладки **Projects**, показанной на рис. 1.1. Создание и использование Custom AppWizard подробно рассматривается в главе 25, *Как добиться повторного использования программных компонентов.*

Мастер Database Project

Если на вашем компьютере установлена редакция Visual C++ Enterprise Edition, можно будет создать проект вида Database Project, орнентированный на работу с базами данных. Этому будет посвящена глава 23, *SQL и редакция Visual C++ Enterprise Edition.*

Мастер DevStudio Add-in Wizard

Под термином *Add-in* (вставка, включенне) понимается своего рода макрос, который автоматизнрует работу Visual Studio, напнсанный на языке C++ или другом языке достаточно высокого уровня, а не на языке VBScript, который нспользуется для "настоящих" макросов. Такие вставки используют средства автоматизацин для маннпулирования Visual Studio.

Мастер ISAPI Extension Wizard

ISAPI — это аббревиатура от *Internet Server API* (Программный интерфейс приложений для Internet-сервера). Под ISAPI понимаются функции, которые можно вызвать для взаимодействия с выполняющейся копией Microsoft Internet Information Server: программой-сервером World Wide Web, которая обслуживает Web-страницы в ответ на запрос пользователя. ISAPI можно использовать для разработки DLL-модулей, которые могут применяться более сложными программами, чем Web-броузер. Связанные с ISAPI вопросы обсуждаются в главе 18.

Мастер Makefile

Этот варнант мастера нужно выбирать в том случае, еслн вы собираетесь создать проект, который будет "собираться" утилнтой Make, отличной от Visual Studio. Если вы не знаете, что это за зверь, — утилита Make, не волнуйтесь: этот мастер предназначен для тех, кто предпочитает использовать автономные инструментальные средства вместо встроенных в Visual Studio.

Мастер MFC ActiveX ControlWizard

Элементы управления ActiveX — это созданные пользователем элементы управления, которые можно нспользовать как в приложеннях, разработанных в среде Visual C++, так и в экранных формах Visual Basic и даже в Web-страницах. Этн элементы управления представляют собой прекрасную 32-разрядную замену для элементов типа VBX, которые используются многими разработчиками для того, чтобы организовать ннтуитивно понятный пользовательский интерфейс или не нзобретать велосипед в новом проекте. Глава 17 послужит вам прекрасным путеводителем по всем закоулкам процесса формирования элементов управления с помощью этого мастера.

Мастер MFC AppWizard (DLL)

Если вы хотите собрать множество функций в DLL-модуль, чтобы все они нспользовали классы нз библиотеки MFC, этот вариант мастера послужит вам прекрасным инструментом. Если же функции не используют MFC, воспользуйтесь вариантом Win32 Dynamic-Link Library, о котором речь пойдет чуть ниже. Создание DLL-модулей освещается в главе 28, *Что еще полезно знать.* AppWizard сформирует для вас текст программы, так что можно будет сразу приступить к делу.

Мастер Win32 Application

Иногда возникает ситуация, когда желательно создать приложеиие на Visual C++, которое не использовало бы библиотеки MFC и тех полуфабрикатов программного кода, которые готовит AppWizard. В таком случае следует выбрать из левой части вкладки **Projects** вариант мастера Win32 Application, указать имя папки для нового проекта и щелкнуть на **OK**. После этого AppWizard молча, не задавая вам ненужных вопросов, создаст файл проекта и откроет его. Теперь вам нужно только написать тексты программ без единой подсказки со стороны AppWizard и включить их в проект.

Мастер Win32 Console Application

Консольное приложение — это, по сути, то же самое, что и знакомое ветеранам DOS-приложение, но выполняется оно в окне регулируемого размера. Оно имеет четко выраженный символьный интерфейс, ориентированный на клавиши управления курсором, а не на мышь. Для программирования пользовательского интерфейса в таких приложениях используется Console API и функции символьного ввода-вывода типа printf()и scanf(). Никаких полуфабрикатов и в этом случае AppWizard для вас не приготовит — только пустой файл проекта. Создание и использование приложений такого вида обсуждается в главе 28.

Мастер Win32 Dynamic-Link Library

Тем, кто планирует разработать DLL-модуль без использования классов из MFC и, таким образом, не нуждается в полуфабрикатах, которые готовит AppWizard, следует предпочесть вариант Win32 Dynamic-Link Library обсуждавшемуся выше варианту MFC AppWizard (DLL). В этом случае вы мгновенно получите пустой файл проекта без всяких неуместных вопросов.

Мастер Win32 Static Library

Хотя в большинстве случаев фрагменты программ, которые используются не одним приложением, собираются в DLL-модули, иногда оказывается предпочтение жесткому подключению всех функций к выполняемому файлу программы. Это избавляет от необходимости распространять совместно с приложением еще и DLL-модули. Выберите этот вариант мастера из списка в левой части окна **New Project Workspace**, и будет создан файл проекта, в который можно будет добавить объектные файлы с тем, чтобы они были скомпонованы в статическую библиотеку, а затем прикомпонованы к выполняемому файлу приложения.

Изменение настройки параметров проекта

AppWizard для некоторого проекта запускается один раз. Предположим, вы создаете типовое приложение. Выберите **File⇨New**, затем — вкладку **Projects**, введите имя проекта и папки, выберите вариант мастера MFC AppWizard (.exe), легко и непринужденно пройдите шесть этапов настройки, после чего будет создан некоторый прототип вашего приложения. Больше никакой необходимости в AppWizard, по крайней мере для данного проекта, у вас вроде бы и нет. Но что если, уже сделав проект наполовину и поразмыслив, вы решите, что отказ, например, от оперативной справки в проекте, был, мягко говоря, поступком опрометчивым?

Нужно сказать, что AppWizard, несмотря на очень звучное название, — все-таки не волшебник. Он включает в файлы проекта те же биты и фрагменты текстов программ, которые могли бы включить и вы сами.

Так что ваши дальнейшие действия довольно прозаичны. Сначала создайте "дубликат" своего первоначального проекта, т.е. закажите AppWizard весь набор функций, который был ранее, в том числе и ту, которую собираетесь изменить. Затем в другой папке создайте проект с тем же именем и теми же настройками, кроме одной — той, которую откорректируйте в соответствии с новыми веяниями. В нашем гипотетическом примере установите флажок Context sensitive Help (Контекстная справка) в окне четвертого этапа настройки. Теперь сравните файлы с помощью утилиты WinDiff, которая имеется в комплекте продукта Visual C++, и узнаете, что нужно изменить в уже побывавшем в работе проекте. С этой добавкой создаваемое приложение будет обладать функцией, которую вначале забыли заказать у AppWizard.

Часто программисты, которые обнаружили просчет на ранней стадии работы над проектом, предпочитают не морочить себе голову сравнением файлов и кодов и заново создают проект с откорректированной настройкой AppWizard и быстренько повторяют уже проделанную с проектом работу или, если это возможно, просто включают в него наработанные тексты программ и ресурсы. В общем, все это дело вкуса и обстоятельств — как далеко вы убежали от настигающего вас паровоза. Единственное, что нужно вынести из подобной ситуации, — в следующий раз формируйте заказ AppWizard, тщательно его продумав, и не торопитесь щелкать на OK в последнем окне.

Текст программы, формируемый AppWizard

В принципе, вас не должен интересовать текст программы, который формирует по вашему заказу AppWizard, особенно если до сих пор вы не писали программ на C++. Для продолжения работы над проектом нет нужды заглядывать в этот полуфабрикат, но, тем не менее, для общего развития мы даем вам некоторое представление о том, что же натворил этот "волшебник". В данном разделе вы познакомитесь с фрагментами текстов программ SDI-приложения, MDI-приложения и простого диалогового приложения.

Если вы самостоятельно не сформировали тех простейших приложений FirstSDI, FirstMDI и FirstDialog, о которых шла речь выше, сейчас самое время их создать. Если вы не очень уверенно чувствуете себя в общении с Visual Studio, загляните в приложение В и обратите основное внимание на такие операции, как редактирование текстов программ и просмотр характеристик классов.

Приложение с единственным документом

SDI-приложение имеет меню, которое пользователь может применить для того, чтобы открыть какой-либо документ (но только один) и затем с ним работать. В этом разделе мы рассмотрим тексты программ, подготовленные для такого приложения AppWizard при следующей настройке: отсутствует поддержка операций с базами данных и составными документами, но есть панель инструментов, строка состояния, оперативная справка и комментарии в тексте программы, функции из библиотеки MFC подключены в режиме разделяемых DLL-модулей. Другими словами, это настройка, которую предлагает AppWizard по умолчанию на всех этапах после первого.

Будет создано пять классов. Имена этих классов для приложения FirstSDI перечислены ниже (обратите внимание, что в некоторых именах классов зримо присутствует имя приложения).

- CAboutDlg — класс диалога для окна About
- CFirstSDIApp — класс для приложения в целом, порожденный CWinApp
- CFirstSDIDoc — класс документа
- CFirstSDIView — класс просмотра
- CMainFrame — класс фрейма окна

Классы диалога рассматриваются в главе 2, *Диалоговые окна и элементы управления*. Классы документа, представления и фрейма окна рассматриваются в главе 4. Файл заголовка (header file) для класса CFirstSDIApp приведен в листинге 1.1. При работе с Visual Studio проще всего познакомиться с содержимым этого файла, щелкнув дважды на имени класса CFirstSDIApp в окне ClassView. После этого текст файла заголовка, соответствующего отмеченному классу, появится в окне редактора кода.

Листинг 1.1. FirstSDI.h — главный файл заголовка для приложения FirstSDI

```
// FirstSDI.h : Главный файл заголовка для приложения FIRSTSDI
//

#if !defined(AFX_FIRSTSDI_H__CDF38D8A_8718_11D0_B02C_0080C81A3AA2__INCLUDED_)
#define AFX_FIRSTSDI_H__CDF38D8A_8718_11D0_B02C_0080C81A3AA2__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#ifndef __AFXWIN_H__
  #error Включение stdafx.h предшествует включению этого файла для PCH
#endif

#include "resource.h"       // Главные символы.

/////////////////////////////////////////////////////////////////////////////
// CFirstSDIApp:
// Использование этих классов приводится в FirstSDI.cpp.
//

class CFirstSDIApp : public CWinApp
{
public:
        CFirstSDIApp();

// Перегрузка.
        // Виртуальная функция, созданная ClassWizard,  перегружает
        //{{AFX_VIRTUAL(CFirstSDIApp)
        public:
        virtual BOOL InitInstance();
        //}}AFX_VIRTUAL

// Реализация.

        //{{AFX_MSG(CFirstSDIApp)
        afx_msg void OnAppAbout();
                // ВНИМАНИЕ!!  Здесь ClassWizard будет добавлять
                // и удалять функции-члены.
                //    НЕ РЕДАКТИРУЙТЕ текст в этих блоках!
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ будет вставлять дополнительные
//    объявления непосредственно
//    перед предыдущей строкой.

#endif // !defined(AFX_FIRSTSDI_H__CDF38D8A_8718_11D0_B02C_0080C81A3AA2__INCLUDED_)
```

Начало текста программы несколько необычно. Директива #if !defined, за которой следует очень длинное выражение (в вашем варианте оно может несколько отличаться), есть в явном виде *защита заголовка* (include guarding). Вам наверняка приходилось встречаться в текстах программ с фрагментами такого типа:

```
#ifndef test_h
#include "test.h"
#define test_h
#endif
```

Подобная конструкция гарантирует, что файл заголовка test.h не будет включен в данный модуль дважды. Двойное включение файла заголовка крайне нежелательно в программах на C++. Предположим, определен класс Employee (персонал) и он использует класс Manager (менеджер). Если оба файла заголовков и для класса Manager, и для класса Employee, в свою очередь, включают файл BigCorp.h, то компилятор выдаст сообщение об ошибке, гласящее, что некоторые символы *переопределены*. А произошло это из-за неявного повторного включения в модуль файла BigCorp.h.

Использование приведенной выше конструкции не решает проблемы. Если некто включит test.h, но забудет определить константу test_h, появится возможность повторно включить файл test.h. Кардинальное решение вопроса состоит в том, чтобы включить и проверку, и определение константы в сам файл заголовка. Тогда фрагмент файла test.h будет выглядеть следующим образом:

```
#ifndef test_h
... собственно текст файла
#define test_h
#endif
```

AppWizard в созданном им файле сформировал более длинное выражение, а не просто test_h. Это длинное выражение предотвращает возникновение проблем с одинаковыми именами файлов заголовков, которые, однако, размещены в разных папках. Кроме того, используется иесколько отличный синтаксис проверки выражения. Директива #pragma once является второй ступенью предохранения от повторного определения объектов в случае, если все-таки кто-то попробует еще раз включить файл заголовка.

По-настоящему "съедобной начинкой" файла заголовка, ради которой и затевался весь этот сыр-бор, является объявление класса CFirstSDIApp. Этот класс — наследник CWinApp, класса MFC, который включает в себя большинство функциональных возможностей, необходимых приложению. AppWizard сгенерировал несколько функций для класса-наследника, которые перегружают соответствующие функции базового класса. Фрагмент текста, который начинается с комментария // Перегрузка, как раз и представляет собой перегрузку виртуальной функции. AppWizard генерирует странный, на первый взгляд, комментарий вокруг объявления перегружаемой виртуальной функции InitInstance(). Эти комментарии затем будут использованы ClassWizard и облегчат включение им при необходимости новых перегрузок. Следующая секция текста программы — карта сообщений; в ней объявляется функция OnAppAbout(). О карте сообщений будет рассказано более подробно в главе 3, *Сообщения и команды*.

AppWizard в файле FirstSDI.cpp генерирует текст функций-членов класса CFirstSDIApp: конструктора, InitInstance()и OnAppAbout(). Ниже приведен листинг конструктора, который инициализирует объект класса.

```
CFirstSDIApp::CFirstSDIApp()
{
 // TODO: сюда добавить текст программы конструктора.
 // Включите все существенные для объекта инициализации в InitInstance.
}
```

Это конструктор, типичный для Microsoft. Поскольку конструктор ничего не возвращает, сообщить программе о каких-то проблемах, возникших при инициализации объекта, совсем не просто. Есть два совершенно различных способа выхода из создавшейся ситуации. Подход, исповедуемый Microsoft, — так называемая двухэтапная инициализация. При этом создается отдельная функция инициализации, так что сам конструктор, по сути, ничего не инициализирует. Эта функция имеет стандартное название `InitInstance` (Инициализировать_экземпляр), а ее текст представлен в листинге 1.2.

Листинг 1.2. `CFirstSDIApp::InitInstance()`

```
BOOL CFirstSDIApp::InitInstance()
{
	AfxEnableControlContainer();

	// Стандартная инициализация.
	// Если вы не будете использовать эту функцию или желаете
	// уменьшить размер выполняемого модуля, удалите те
	// из следующих ниже подпрограмм инициализации, в которых
	// нет необходимости.

#ifdef _AFXDLL
	Enable3dControls();        // Эту функцию следует вызывать в том
	                           // случае, если используются DLL-модули MFC.
#else
	Enable3dControlsStatic();        // Эту функцию следует вызывать
	                           // в том случае, если функции из MFC
	                           // прикомпоновываются статически.
#endif

	// Измените ключ регистрации, под которым хранятся установки.
	// Эту строку нужно изменить соответственно наименованию
	//  вашей фирмы или организации.
	SetRegistryKey(_T("Local AppWizard-Generated Applications"));

	LoadStdProfileSettings();  // Загрузить стандартный файл
	                                   //  опций INI (включая MRU).

	// Зарегистрировать шаблон документа. Шаблон документа
	//  служит в качестве связующего звена между документами,
	//  фреймом окна и представлениями.

	CSingleDocTemplate* pDocTemplate;
	pDocTemplate = new CSingleDocTemplate(
		IDR_MAINFRAME,
		RUNTIME_CLASS(CFirstSDIDoc),
		RUNTIME_CLASS(CMainFrame),       // Фрейм главного SDI-окна.
		RUNTIME_CLASS(CFirstSDIView));
	AddDocTemplate(pDocTemplate);

	// Разбиение командной строки для команд оболочки, DDE
	//  и открытия файлов.
	CCommandLineInfo cmdInfo;
	ParseCommandLine(cmdInfo);

	// Распределить функции, заданные в командной строке.
	if (!ProcessShellCommand(cmdInfo))
		return FALSE;

	// Создается первое и единственное окно программы;
	//  показать и обновить его.
```

```
    m_pMainWnd->ShowWindow(SW_SHOW);
    m_pMainWnd->UpdateWindow();

    return TRUE;
}
```

Функция InitInstance()готовит приложение к работе. Все начинается с разрешения приложению содержать в своем составе элементы управления ActiveX, для чего вызывается AfxEnableControlContainer(). Затем наступает очередь объемного дизайна элементов управления и ключа регистрации приложения. (Регистрация описывается в главе 7. Если вы с ней еще не знакомы, можете сейчас не забивать себе голову ее назначением.)

Следующее действие InitInstance() — регистрация единственного шаблона документа, того самого, который будет создан SDI-приложением. Документы, средства просмотра, фреймы окон и шаблоны документов будут рассмотрены позже, в главе 4.

Для анализа командной строки InitInstance() организует пустой объект класса CCommandLineInfo. Затем функция ParseCommandLine()помещает в него параметры, заданные в командной строке при запуске приложения. И наконец, вызывается ProcessShellCommand(), которая должна организовать выполнение операций, заданных этими параметрами. Это означает, что ваше приложение сможет поддерживать параметры командной строки и дать, таким образом, возможность пользователю сэкономить время, необходимое для настройки его работы. Причем, что самое интересное, от вас — разработчика — не требуется для этого даже шевелить пальцами. Например, если пользователь наберет в командной строке FirstSDI fooble, приложение сразу же после вызова откроет файл fooble. Функция ProcessShellCommand()поддерживает следующие параметры в командной строке.

Параметр	Назначение
без параметров	Запускает приложение и открывает новый файл
Имя_файла	Запускает приложение и открывает указанный файл
/p Имя_файла	Запускает приложение и распечатывает указанный файл на принтере, заданном по умолчанию
/pt Имя_файла Принтер Драйвер Порт	Запускает приложение и распечатывает указанный файл на указанном принтере
/dde	Запускает приложение и ждет команды DDE (динамического обмена данными)
/Automation	Запускает приложение в качестве автоматного сервера OLE
/Embedding	Запускает приложение в режиме редактирования внедренного объекта OLE

Если вам понадобится реализовать какие-либо другие функции, создайте класс, который унаследует от CCommandLineInfo способность накапливать компоненты, образующиеся при анализе командиой строки, а затем в собственном App-классе перегрузите функции CWinApp::ParseCommandLine() и CWinApp::ProcessShellCommand().



Возможно, вам уже известно, что многие программы Windows можно вызывать из командной строки. Например, если набрать на клааиатуре Notepad blah.txt, то будет открыт файл blah txt в программе Notepad. Также можно обращаться и к другим параметрам командной строки. Набрав Notepad /p blah.txt, вы откроете файл blah.txt в программе Notepad и распечатаете его.

Последний оператор в InitInstance() возвращает TRUE, извещая таким образом вызывающую программу о том, что инициализация завершена и можно приступать к работе.

Присутствие карты сообщений в файле заголовка является индикатором того, что функция OnAppAbout организует вывод сообщения. Какого именно? Вот как выглядит карта сообщений в файле текста программы.

```
BEGIN_MESSAGE_MAP(CFirstSDIApp, CWinApp)
	//{{AFX_MSG_MAP(CFirstSDIApp)
	ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
		// ВНИМАНИЕ! Здесь ClassWizard будет вставлять и
		//   удалять макросы отображения сообщений.
		//   НЕ РЕДАКТИРУЙТЕ текст в этом блоке!
	//}}AFX_MSG_MAP
	// Стандартные команды работы с файлами документов.
	ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
	ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
	// Стандартные команды установки принтера.
	ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()
```

Эта карта сообщений перехватывает команды из меню, о чем будет подробно рассказано в главе 3. Когда пользователь выберет Help⇨About, будет вызвана функция-член CFirstSDIApp::OnAppAbout(). Если же пользователь выберет File⇨New, File⇨Open или File⇨Print Setup, то за дело возьмутся соответствующие функции-члены класса CWinApp. Эти функции можно перегрузить функциями собственной разработки, если вы хотите, чтобы они выполняли нечто нестандартное в ответ на перечисленные команды меню. Текст функции OnAppAbout() выглядит следующим образом:

```
void CFirstSDIApp::OnAppAbout()
{
	CAboutDlg aboutDlg;
	aboutDlg.DoModal();
}
```

Здесь объявляется объект, который является экземпляром класса CAboutDlg, и вызывается функция DoModal(), которая выводит на экран окно диалога. И диалоговые классы, и функция DoModal() рассматриваются подробно в главе 2. Изменять что-либо в стандартной процедуре обработки щелчков на кнопках OK и Cancel нет никакой необходимости — а это все, что делается в простейшем окне сообщения.

Другие файлы

Если при настройке AppWizard вы выбрали опцию Context sensitive Help (Контекстная справка), то для реализации этой возможности в приложении будет сформирован файл .HPJ и несколько файлов .RTF. Подробно об этих файлах будет рассказано в главе 11, в разделе *Компоненты справочной системы*.

Кроме того, AppWizard формирует файл README.TXT, в котором поясняется, какие сформированы файлы и какие новые классы созданы. Желательно прочесть этот файл, особенно если что-нибудь в очень похожих по названию файлах вас смущает[3].

Создается также несколько файлов, хранящих информацию о настройке среды разработки для данного проекта. Эти файлы ускоряют процесс компиляции и компоновки разрабатываемого приложения, хранят промежуточную информацию компилятора, касающуюся переменных и функций в приложении. Такие файлы имеют расширения .ncb, .aps, .dsw и т.д. Как правило, разработчику никогда не приходится работать с подобными файлами напрямую — они нужны только среде разработки.

---

[3] Естественно, AppWizard готовит текст файла README.TXT на *английском* языке. — *Прим. ред.*

Содержимое MDI-приложения

MDI-приложение имеет меню, но позволяет пользователю одновременно работать более чем с одним документом. В этом разделе будут рассмотрены тексты программ, которые готовит AppWizard при выборе такого вида приложения. При этом полагается, что была сделана следующая настройка: отсутствует поддержка операций с базами данных и составными документами, но есть панель инструментов и строка состояния, оперативная справка, комментарии в тексте программы, функции из библиотеки MFC подключены в виде DLL-модулей. Как и в случае SDI-приложения, это настройка, которую предлагает AppWizard по умолчанию на всех этапах после первого. Основное внимание в этом разделе будет уделено отличиям в тексте программ от рассмотренного в предыдущем разделе SDI-приложения.

Будет создано пять классов. Имена этих классов для приложения FirstMDI перечислены ниже.

- CAboutDlg — класс диалога для окна About
- CFirstMDIApp — класс для приложения в целом, порожденный CWinApp
- CFirstMDIDoc — класс документа
- CFirstMDIView — класс представления
- CMainFrame — класс фрейма окна

Заголовок App-класса приложения представлен ниже, в листинге 1.3.

Листинг 1.3. FirstMDI.h — главный файл заголовка для приложения FirstMDI

```
// FirstMDI.h : Главный файл заголовка для приложения FIRSTMDI
//

#if !defined(AFX_FIRSTMDI_H__CDF38D9E_8718_11D0_B02C_0080C81A3AA2__INCLUDED_)
#define AFX_FIRSTMDI_H__CDF38D9E_8718_11D0_B02C_0080C81A3AA2__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#ifndef __AFXWIN_H__
  #error Включение stdafx.h предшествует включению этого файла для PCH
#endif

#include "resource.h"        // Главные символы.

/////////////////////////////////////////////////////////////////////////////
// CFirstMDIApp:
// Использование этих классов приводится в FirstMDI.cpp.
//

class CFirstMDIApp : public CWinApp
{
public:
        CFirstMDIApp();

// Перегрузка.
        // Виртуальная функция, созданная ClassWizard, перегружает
        //{{AFX_VIRTUAL(CFirstMDIApp).
        public:
        virtual BOOL InitInstance();
        //}}AFX_VIRTUAL
```

```
// Реализация.

        //{{AFX_MSG(CFirstMDIApp)
        afx_msg void OnAppAbout();
                // ВНИМАНИЕ!!  Здесь ClassWizard будет добавлять и
                //'  удалять функции-члены.
                //   НЕ РЕДАКТИРУЙТЕ текст в этих блоках!
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ будет вставлять дополнительные
// объявления непосредственно перед предыдущей строкой.

#endif // !defined(AFX_FIRSTMDI_H__CDF38D9E_8718_11D0_B02C_0080C81A3AA2_
➥_INCLUDED_)
```

В чем же отличие этого файла от FirstSDI.h? Только в именах классов. Даже конструктор тот же самый. Функция OnAppAbout() в точности повторяет вариант для SDI-приложения. А как насчет InitInstance()? Текст этой функции приведен в листинге 1.4.

Листинг 1.4. CFirstMDIApp::InitInstance()

```
BOOL CFirstMDIApp::InitInstance()
{
        AfxEnableControlContainer();

        // Стандартная инициализация.
        // Если вы не будете использовать эту функцию или желаете
        //  уменьшить размер выполняемого модуля, удалите те из
        //  следующих ниже подпрограмм инициализации, в которых нет
        //  необходимости.

#ifdef _AFXDLL
        Enable3dControls();           // Эту функцию следует вызывать в том
                                      // случае, если используются DLL-модули MFC.
#else
        Enable3dControlsStatic();          // Эту функцию следует вызывать
                // в том случае, если функции из MFC прикомпоновываются
                // статически.
#endif

        // Измените ключ регистрации, под которым хранятся установки.
        // Эту строку нужно изменить соответственно наименованию
        //  вашей фирмы или организации.
        SetRegistryKey(_T("Local AppWizard-Generated Applications"));

        LoadStdProfileSettings();  // Загрузить стандартный файл
                          // опций INI (включая MRU).

        // Зарегистрировать шаблон документа. Шаблон документа
        //  служит в качестве связующего звена между документами,
        // фреймом окна и представлениями.

        CMultiDocTemplate* pDocTemplate;
        pDocTemplate = new CMultiDocTemplate(
```

```
            IDR_FIRSTMTYPE,
            RUNTIME_CLASS(CFirstMDIDoc),
            RUNTIME_CLASS(CChildFrame), // Фрейм дочерних окон MDI.
            RUNTIME_CLASS(CFirstMDIView));
    AddDocTemplate(pDocTemplate);

    // Создать фрейм главного MDI-окна.
    CMainFrame* pMainFrame = new CMainFrame;
    if (!pMainFrame->LoadFrame(IDR_MAINFRAME))
            return FALSE;
    m_pMainWnd = pMainFrame;

    // Разбиение командной строки для команд оболочки, DDE
    //  и открытия файлов.
    CCommandLineInfo cmdInfo;
    ParseCommandLine(cmdInfo);

    // Распределить функции, заданные в командной строке.
    if (!ProcessShellCommand(cmdInfo))
            return FALSE;

    // Главное окно программы инициализировано, так что можно
    //         показать и обновить его.
    pMainFrame->ShowWindow(m_nCmdShow);
    pMainFrame->UpdateWindow();

    return TRUE;
}
```

В чем же разница? Здесь вам поможет WinDiff — средство, которое входит в состав Visual C++ и вызвать которое можно из меню Tools (Инструменты). Используя WinDiff, сравните версии функции InitInstance()в файлах программ приложений FirstSDI и FirstMDI. Вы увидите, что, кроме имен классов, имеются следующие отличия.

- MDI-приложение формирует экземпляр класса CMultiDocTemplate, а SDI-приложение — класса CSingleDocTemplate. Подробно об этом будет рассказано в главе 4.
- MDI-приложение организует главное окно и затем выводит его на экран, а SDI-приложение этого не делает.

Это сравнение дает ясное представление о преимуществах парадигмы *Документ/Представление* — достаточно серьезное отличие в функционировании приложения достигается небольшими фрагментами текстов программ проекта, причем насколько это возможно все подробности их реализации скрыты от разработчика.

Простое диалоговое приложение

Простое диалоговое приложение значительно проще, чем SDI- и MDI-приложения. Создадим приложение этого вида, FirstDialog, задав следующие параметры AppWizard: имеется окно About, отсутствует оперативная справка, используется объемный дизайн окна, отсутствует поддержка операций с автоматными серверами, но поддерживается работа элементов управления ActiveX, включаются комментарии в тексты программ, функции из библиотеки MFC подключаются к приложению в качестве DLL-модулей. Другими словами, вы не возражаете против тех настроек, которые AppWizard предлагает по умолчанию для приложения такого вида.

Будет создано три класса. Имена этих классов для приложения FirstDialog перечислены ниже.

- CAboutDlg — класс диалога для окна About
- CFirstDialogApp — класс для приложения в целом, порожденный CWinApp
- CFirstDialogDlg — класс диалога для приложения в целом

Классы диалога будут подробно рассмотрены в главе 2. Файл заголовка для приложения FirstDialog представлен ниже, в листинге 1.5.

Листинг 1.5. FirstDialog.h — главный файл заголовка для приложения FirstDialog

```
// FirstDialog.h : Главный файл заголовка для приложения FIRSTDIALOG.
//

#if !defined(AFX_FIRSTDIALOG_H__CDF38DB4_8718_11D0_B02C_0080C81A3AA2__INCLUDED_)
#define AFX_FIRSTDIALOG_H__CDF38DB4_8718_11D0_B02C_0080C81A3AA2__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#ifndef __AFXWIN_H__
  #error Включение stdafx.h предшествует включению этого файла для PCH.
#endif

#include "resource.h"              // Главные символы.

/////////////////////////////////////////////////////////////////////////////
// CFirstDialogApp:
// Использование этих классов приводится в FirstDialog.cpp.

//

class CFirstDialogApp : public CWinApp
{
public:
        CFirstDialogApp();

// Перегрузка.
        // Виртуальная функция, созданная ClassWizard,  перегружает
        //{{AFX_VIRTUAL(CFirstDialogApp).
        public:
        virtual BOOL InitInstance();
        //}}AFX_VIRTUAL

// Реализация.

        //{{AFX_MSG(CFirstDialogApp)
                // ВНИМАНИЕ!!  Здесь ClassWizard будет добавлять и
                //    удалять функции-члены.
                //    НЕ РЕДАКТИРУЙТЕ текст в этих блоках!
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ будет вставлять дополнительные
//      объявления непосредственно перед предыдущей строкой.

#endif //
!defined(AFX_FIRSTDIALOG_H__CDF38DB4_8718_11D0_B02C_0080C81A3AA2__INCLUDED_)
```

Класс CFirstDialogApp является наследником CWinApp, класса MFC, который включает в себя большинство функциональных возможностей, необходимых приложению. Класс CWinApp имеет конструктор, который ничего не делает, как, впрочем, и конструкторы в SDI- и MDI-приложениях, рассмотренные в предшествующих разделах. Кроме того, перегружается функция InitInstance(), как это видно из листинга 1.6.

Листинг 1.6. FirstDialog.cpp — CFirstDialogApp::InitInstance()

```
BOOL CFirstDialogApp::InitInstance()
{
        AfxEnableControlContainer();

        // Стандартная инициализация.
        // Если вы не будете использовать эту функцию или желаете уменьшить
        //  размер выполняемого модуля, удалите те из следующих ниже
        //  подпрограмм инициализации, в которых нет необходимости.

#ifdef _AFXDLL
        Enable3dControls();       // Эту функцию следует вызывать в том
                                  // случае, если используются DLL-модули MFC.
#else
        Enable3dControlsStatic();        // Эту функцию следует вызывать
                // в том случае, если функции из MFC прикомпоновываются
                // статически.
#endif

        CFirstDialogDlg dlg;
        m_pMainWnd = &dlg;
        int nResponse = dlg.DoModal();
        if (nResponse == IDOK)
        {
      // TODO: сюда добавить текст программы, которая будет
                //   обрабатывать щелчок на OK при работе с окном.
        }
        else if (nResponse == IDCANCEL)
        {
                // TODO: сюда добавить текст программы, которая будет
                //  обрабатывать щелчок на Cancel при работе с окном.
        }

        // Поскольку окно закрывается, функция возвращает FALSE.
        // В результате выполнение приложения завершается.
        // В противном случае запустился бы механизм обработки сообщений.
        return FALSE;
}
```

Сначала разрешается режим объемного дизайна элементов управления, поскольку так было задано при настройке AppWizard. Затем выводится диалоговое окно, которое и является окном приложения. Для того чтобы это сделать, функция создает dlg — экземпляр класса CFirstDialogDlg, а затем вызывает функцию DoModal(), которая выводит окно на экран и возвращает IDOK, если пользователь щелкнет на OK, или IDCANCEL, если пользователь щелкнет на Cancel. (Этот процесс будет рассмотрен позже, в главе 2.) Ваша задача состоит в том, чтобы организовать какую-то полезную работу в этом диалоговом окне. Далее функция InitInstance() возвращает FALSE, поскольку мы имеем дело с простым диалоговым приложением, которое прекращает функционировать после того, как диалоговое окно будет закрыто. Если вы обратили внимание, прежде в SDI- и MDI-приложениях функция InitInstance() возвращала TRUE, что трактуется следующим образом: *все хорошо, можно выполнять оставшуюся часть приложения*. В то же время возврат FALSE означал следующее: *что-то у них там не связалось при инициализации.*

Поскольку в данном случае никакого продолжения последовать не должно, в простом диалоговом приложении функция InitInstance() всегда возвращает FALSE.

Обзор настроек AppWizard и глав книги

AppWizard — это настоящий *почемучка*, задающий в ходе настройки довольно много вопросов. В результате он сразу открывает перед вами целый веер дорог, дорожек и троп. В этой главе мы рассмотрели только одну функцию — InitInstance() — и уже на примере простейших приложений показали связь между настройками AppWizard и фрагментами текста программы (в случаях, когда AppWizard формирует простейшее диалоговое приложение, SDI- или MDI-приложение). Большинство других настроек AppWizard нуждается в более серьезном описании. Многим вопросам мы посвятим отдельные главы. Ниже приводится сводная таблица, в которой суммированы возможные настройки и названия глав, посвященных тем или иным вопросам.

Этап	Настройка	Глава	Диалоговое приложение
0	MFC DLL или не MFC DLL	28, Что еще полезно знать	Нет
0	Элементы управления OCX	17, Создание элемента управления ActiveX	—
0	Консольное приложение	28, Что еще полезно знать	—
0	Собственный AppWizard	25, Как добиться повторного использования программных компонентов	—
0	Мастер расширения ISAPI (ISAPI Extension Wizard)	18, Windows Sockets, MAPI и Internet	—
1	Поддержка языка	28, Что еще полезно знать	Да
2	Поддержка баз данных	22, Доступ к базам данных	—
3	Контейнер составных документов	14, Создание приложения-контейнера ActiveX	—
3	Мини-сервер составных документов	15, Создание приложения-сервера ActiveX	—
3	Полный сервер составных документов	15, Создание приложения-сервера ActiveX	—
3	Составные файлы	14, Создание приложения-контейнера ActiveX	—
3	Автоматы	16, Создание сервера автоматизации	Да
3	Использование элементов управления ActiveX	17, Создание элемента управления ActiveX	Да
4	Панели инструментов	9, Панели инструментов и строка состояния	—
4	Панели инструментов	9, Панели инструментов и строка состояния	—
4	Строка состояния	9, Панели инструментов и строка состояния	—
4	Распечатка	6, Распечатка и предварительный просмотр	—
4	Контекстная справка	11, Справка в приложении	Да

Этап	Настройка	Глава	Диалоговое приложение
4	Объемный дизайн	—	Да
4	MAPI	18, Windows Sockets, MAPI и Internet	—
4	Интерфейс Window Sockets	18, Windows Sockets, MAPI и Internet	Да
4	Файлы в списке самых "свежих"	—	—
5	Комментарии в тексте программы	—	Да
5	Библиотека MFC	—	Да
6	Базовые классы представления	4, Документы и представления	—

Поскольку некоторые из этих настроек при создании простого диалогового приложения не встречаются, строки таблицы, соответствующие доступным настройкам, помечены *Да* в колонке *Диалоговое приложение*. Прочерк в колонке *Глава* означает, что назначение и последствия использования данной настройки настолько очевидны, что нет смысла специально "растекаться мыслию по древу". О них упомянуто мимоходом в этой и других главах.

Итак, вы познакомились с азами технологии создания приложений, которые сами по себе ничего полезного сделать не могут. Для того чтобы заставить их выполнять нечто полезное, нужно, как минимум, оснастить их меню и элементами управления диалогом, которые смогут передать программе команды и информацию от пользователя. Это будет рассмотрено в следующей главе, *Диалоговые окна и элементы управления*.

ГЛАВА 2

Диалоговые окна и элементы управления

В этой главе...

Что такое диалоговое окно

Формирование ресурсов диалогового окна

Создание класса диалогового окна

Использование класса диалогового окна

Что такое диалоговое окно

Программы, работающие в среде Windows, имеют *графический интерфейс пользователя* (GUI — graphical user interface). Во времена царствования DOS программа выводила на экран текстовый запрос-приглашение, в котором пользователю предлагалось ввести те или иные данные, в которых в текущий момент нуждалась программа. В операционной среде Windows получить данные от пользователя не так просто, и большая часть информации поступает в программу через *диалоговые окна* (dialog box). Например, пользователь может сообщить приложению о деталях запроса к системе, набрав некоторый текст в *текстовом поле* (text box), сделав выбор из *списка* (list box), включив один из *переключателей* (radio button), установив *флажок* (check box) или выполнив некоторое действие с другими *элементами управления*[4].

Вполне вероятно, что создаваемое вами Windows-приложение будет иметь не одно диалоговое окно, каждое — для общения с пользователем в специфической ситуации. Для каждого диалогового окна в приложении есть две вещи, которые нужно разработать, — *ресурсы* окна и *класс* окна.

Ресурсы окна используются программой для того, чтобы вывести на экран его изображение и изображения элементов управления, которые входят в него. В класс окна включены его параметры и функции-члены, ответственные за вывод окна на экран. Они работают совместно для достижения общей цели — обеспечить максимально эффективное взаимодействие пользователя и программы.

Ресурсы диалогового окна создаются посредством *редактора ресурсов*, с помощью которого вы можете включать в состав окна необходимые элементы управления и размещать их в пространстве окна желаемым образом. Помощь в создании класса окна вам окажет `ClassWizard`. Как правило, класс конкретного диалогового окна в проекте является производным от базового класса `CDialog`, имеющегося в составе MFC. `ClassWizard` также поможет связать ресурсы окна с классом. Обычно каждый элемент управления, включенный в состав ресурсов окна, имеет в классе окна соответствующий член-переменную. Для того чтобы вывести диалоговое окно на экран, нужно вызвать функцию-член его класса. Для того чтобы установить значения по умолчанию для элементов управления перед выводом окна на экран или считать состояние элементов управления после завершения работы пользователя, нужно обращаться к членам-переменным класса.

Формирование ресурсов диалогового окна

Первый шаг процесса организации диалогового окна в приложении, использующем библиотеку MFC, — формирование ресурса окна, который служит своего рода шаблоном для Windows. Когда Windows видит ресурс окна в программе, она использует команды из этого ресурса для конструирования работающего окна.

В этой главе для иллюстрации работы с диалоговыми окнами используется простенькое приложение, в которое будет добавлено диалоговое окно. В качестве такового используется SDI-приложение `FirstSDI`, сформированное на жестком диске компьютера в соответствии с инструкцией, описанной в главе 1. В дальнейшем вы сформируете ресурсы и класс диалогового

---

[4] В англоязычной литературе наименования практически всех средств общения с пользователем (как простых, так и сложных, состоящих из нескольких простых в качестве компонентов) включают слово *box* — "прямоугольник", "коробка". В русскоязычной литературе для сложных средств, состоящих из более чем одного простого, мы применяем слово *окно* в качестве основного компонента наименования типа, а для простых средств — индивидуальные названия без классообразующего слова или словосочетания. — *Прим. ред.*

окна для этого приложения, напишете текст программы, которая отвечает за отображение этого окна на экране, и тексты программ, обрабатывающих данные, введенные пользователем.

Для того чтобы приступить к формированию ресурсов, необходимо сначала открыть приложение. Выберите Insert⇨Resource из меню Visual Studio. Появится диалоговое окно Insert Resource, показанное на рис. 2.1. Дважды щелкните на элементе Dialog в окне Resource Type. Этим вы вызываете редактор диалогового окна, который выводит на экран заготовку окна, как это показано на рис. 2.2.

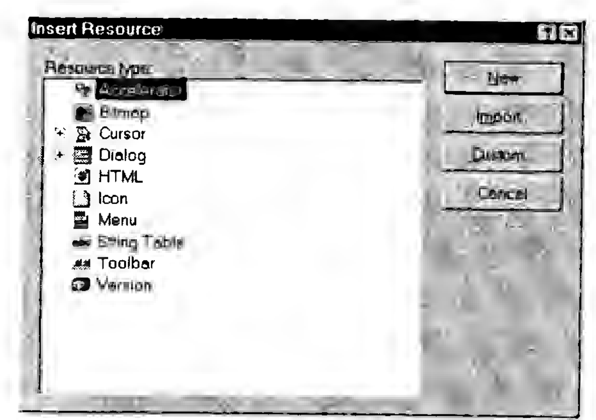
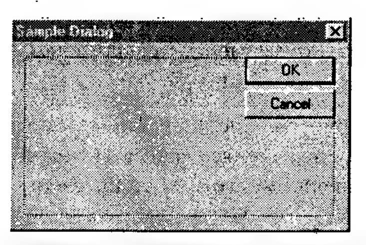


Рис. 2.1. Дважды щелкните на элементе Dialog *в поле* Resource type *диалогового окна* Insert Resource

Рис. 2.2. Заготовка диалогового окна имеет строку заголовка и две кнопки — OK и Cancel

Вызовите на экран диалоговое окно Dialog Properties для вновь создаваемого диалогового окна, выбрав View⇨Properties. В поле Caption (Надпись) введите `Sample Dialog` (Простой диалог), как это показано на рис. 2.3. Вам придется довольно часто обращаться к окну Dialog Properties по ходу разработки ресурсов диалогового окна, так что "закрепите" его на экране, щелкнув на пиктограмме с изображением канцелярской кнопки (такой, какой пользуются в штате Калифорния) в левом верхнем углу.



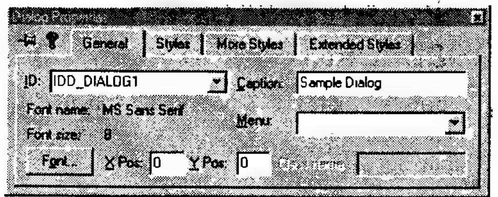


Рис. 2.3. Используйте диалоговое окно Dialog Properties *для изменения заголовка вновь создаваемого диалогового окна*

Богатая палитра образцов элементов управления (инструментарий) имеется в вашем распоряжении для того, чтобы вставлять те или иные из них в диалоговое окно приложения. Для установки элементов диалогового окна используется технология, получившая наименование *WYSIWYG*[5]. Чтобы установить в своем диалоговом окне кнопку, нужно выбрать ее образец на поле инструментария, "перетянуть" в желаемое место на поле заготовки окна приложения, "сбросить" ее там и заменить надпись — вместо `Button1` задать `Lookup` (Просмотр) или `Connect` (Подсоединить), или что вы еще там придумаете. В диалоговом окне приложения вы можете использовать все типовые для приложений Windows элементы управления.

- **Надпись** (Static text). По существу, это "неполноценный" элемент управления, поскольку он используется только как поле для вывода надписи, относящейся к "настоящему" элементу управления, расположенному по соседству.
- **Текстовое поле** (Edit box). Текстовое поле может быть однострочным или многострочным; сюда пользователь может ввести текст или число — последовательность цифр — в качестве данных для программы.
- **Кнопка** (Button). В заготовке каждого диалогового окна уже присутствуют кнопки OK и Cancel, но можно добавить еще и свои — столько, сколько посчитаете нужным.
- **Флажок** (Check box). Флажки используются для установки опций, каждая из которых может быть выбрана независимо от других.
- **Переключатель (радиокнопка)** (Radio button). Эти элементы управления используются для выбора одной из групп связанных опций; если выбрана одна из них, то другие полагаются невыбранными.

---

[5] Это аббревиатура от What You See Is What You Get — Что видишь, то и получишь. — *Прим. ред.*

- **Список** (List box). Элемент этого типа используется для выбора одного элемента из заранее подготовленного набора; набор может быть как жестко установленным на этапе разработки программы, так и меняться программно в процессе выполнения приложения; главное — пользователь по своей воле не может непосредственно менять элементы в наборе; он может только их выбирать.
- **Поле со списком** (Combo box). Это комбинация текстового поля и списка; такой элемент управления позволяет пользователю не только выбирать элементы из ранее подготовленного набора, но и самостоятельно пополнять его, непосредственно "впечатывая" необходимый текст в текстовое поле.

Простое приложение, которое будет создано в этой главе, будет оперировать диалоговым окном с рядом элементов управления, так что вы сможете освоить технологию включения элементов разных типов в диалоговое окно приложения.

Задание идентификаторов диалогового окна и элементов управления

Поскольку каждое диалоговое окно в приложении является уникальным объектом (исключение составляют только стандартные окна, о которых речь будет идти в последующих главах), разработчику практически всегда нужно присваивать окнам и элементам управления, входящим в их состав, идентификаторы по собственному выбору. Конечно, можно согласиться и с теми идентификаторами, которые предлагает редактор диалоговых окон по умолчанию. Однако эти имена тривиальны (как правило, нечто вроде IDD_DIALOG1, IDC_EDIT1, IDC_RADIO1), и значительно лучше заменить их другими, связанными с назначением и функциями окна или элемента. Но в любом случае рекомендуется соблюдать соглашение о префиксах: идентификаторы диалоговых окон имеют префикс IDD_, а идентификаторы элементов управления — IDC_. Заменить идентификатор можно с помощью диалогового окна Dialog Properties. Для этого выберите элемент управления — щелкните на нем (или на свободном поле окна для выбора всего окна) и выберите Edit⇨Properties, если ранее окно Dialog Properties не было выведено и закреплено на экране. Затем измените идентификатор ресурса в поле ID на нечто осмысленное, но при этом не забывайте о префиксах: IDD_ — для диалоговых окон и IDC_ — для элементов управления.

Создание диалогового окна Sample Dialog

Щелкните на пиктограмме текстового поля в инструментарии, а затем щелкните в левом верхнем углу поля заготовки диалогового окна (в том месте, где будет расположен этот элемент). Если посчитаете нужным, "ухватитесь" указателем мыши за один из размерных маркеров вокруг изображения элемента и просто перетащите его на то место, которое он занимает на рис. 2.4. Как уже было сказано выше, вы можете изменить идентификатор элемента, заданный по умолчанию, но для данного случая это необязательно. Поэтому не будем отвлекаться на оформление программы.

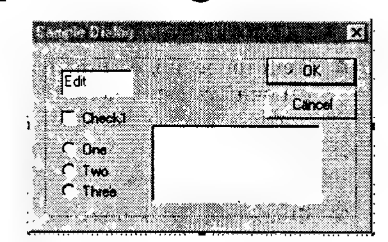


Рис. 2.4. Диалоговое окно можно быстро сформировать, используя редактор ресурсов



Пока вы не научитесь работать с инструментарием элементов управления, вполне вероятно, что вы будете путаться в том множестве пиктограмм, которые в нем имеются. Помощь вам окажет *контекстное окно указателя* (Tooltip). Если установить указатель мыши на некоторой пиктограмме и задержать его там на непродолжительное время, то рядом с указателем откроется маленькое окошко, в котором будет выведена надпись — наименование соответствующего элемента[6]. Перемещая указатель от одной пиктограммы к другой, найдите нужный вам элемент.

Добавьте элемент типа флажок и три переключателя в диалоговое окно, чтобы оно приняло вид, изображенный на рис. 2.4. В поле Caption измените надписи для переключателей. Пусть это будут One (Один), Two (Два) и Three (Три). Чтобы подравнять включенные в окно элементы (выстроить их в колонку), щелкните на одном из них, а затем, нажав и удерживая клавишу <Ctrl>, щелкайте по очереди на остальных. После этого выберите Layout⇨Align Control⇨Left (Размещение⇨Подравнять элементы⇨Слева). После этого, если возникнет желание, можно их все вместе сместить, "перетягивая" мышью. Делать это можно до тех пор, пока все элементы полагаются вместе выбранными. Затем выберите команду Layout⇨Space Evenly⇨Down (Размещение⇨Подравнять интервал⇨Вниз). Эта команда позволяет установить одинаковый интервал между элементами по вертикали.



Команды меню Layout дублируются на панели инструментов Dialog, которая появляется в нижней части экрана, как только вы обращаетесь к услугам редактора ресурсов. Пиктограммы на панели инструментов те же, что и в окнах меню. Это поможет вам быстрее запомнить связь между командами меню и пиктограммами панели.

Щелкните еще раз на переключателе One и вызовите окно Dialog Properties. Установите в нем флажок Group (Группа). Такая установка означает, что переключатель One является первым элементом группы переключателей, т.е. именно к переключателям, объединенным в группу, будет в дальнейшем применяться принцип "только один прав".

Теперь добавьте в формируемое окно еще и элемент типа список. Установите его справа от переключателей и измените размеры таким образом, чтобы ваше творение и то, что изображено на рис. 2.4, стали походить друг на друга, как близнецы-братья. В то время, когда этот элемент еще остается выбранным, с помощью команды View⇨Properties вызовите на экран окно Dialog Properties. Выберите вкладку Styles (Стили) и проверьте, не установлен ли флажок Sort (Сортировка). Если этот флажок установлен, то при выполнении программы элементы в списке будут отсортированы по алфавиту. Для нашего приложения это излишняя роскошь. Здесь элементы должны размещаться в списке в том порядке, в котором мы их будем включать.

Создание класса диалогового окна

Когда формирование ресурсов диалогового окна будет завершено, вызовите на экран диалоговое окно мастера ClassWizard. Для этого нужно выбрать View⇨ClassWizard. Мастер ClassWizard сам разберется в ситуации — создано новое диалоговое окно, но класс для него не определен — и предложит свои услуги, как это показано на рис. 2.5. Не трогайте установленный переключатель Create a new class (Создать новый класс), а сразу щелкните на OK. Появится новое диалоговое окно New Class (Новый класс), которое показано на рис. 2.6. В поле Name (Имя) введите CSdiDialog (имя нового класса) и щелкните на OK. После этого ClassWizard создаст новый класс, подготовит файл текста программы SdiDialog.cpp и файл заголовка SdiDialog.h и включит их в состав проекта.

---

[6] Естественно, в нелокализованной версии будет выведено английское наименование. Поэтому в приведенном выше перечне мы возле каждого русского названия элемента управления привели его английский эквивалент. — *Прим. ред.*

Вкладка Member Variables, показанная на рис. 2.7, позволяет легко установить соответствие между текстом программы и ресурсами окна. Щелкните в строке IDC_CHECK1, а затем на кнопке Add Variable. Это приведет к появлению на экране диалогового окна Add Member Variable, показанного на рис. 2.8.

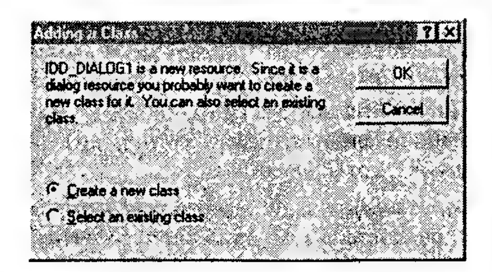


Рис. 2.5. ClassWizard предлагает вам создать класс для нового диалогового окна. Надпись гласит: IDD_DIALOG1 – это новый ресурс. Поскольку это ресурс диалогового окна, вы, вероятно, не против создать для него и новый класс. Но, если хотите, можете использовать уже существующий класс

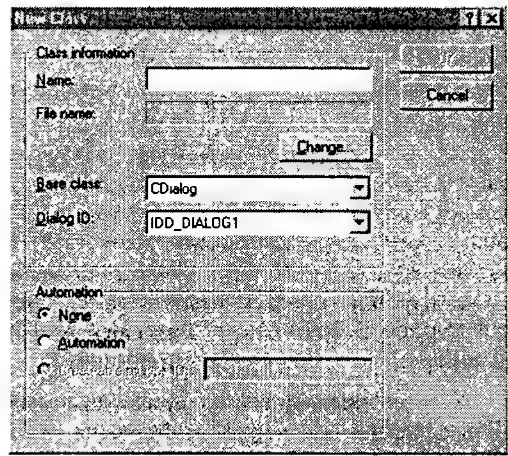


Рис. 2.6. С помощью ClassWizard создание класса для диалогового окна выполняется очень просто

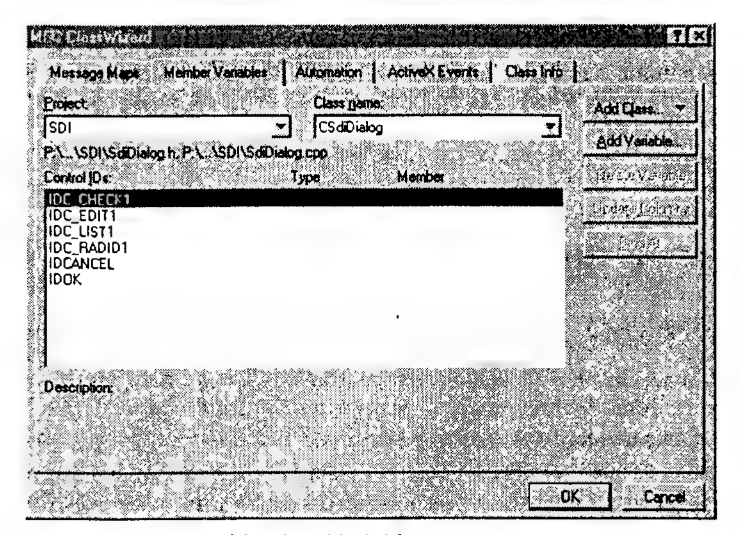


Рис. 2.7. Вкладка Member Variables позволяет установить соответствие между членами-переменными класса и элементами управления, включенными в диалоговое окно

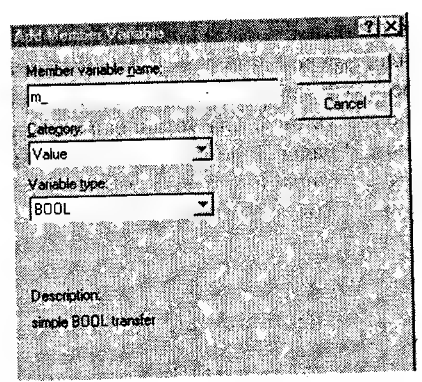


Рис. 2.8. Установка идентификатора члена-переменной класса, ассоциированного с некоторым элементом управления

Член-переменная класса нового диалогового окна соответствует либо значению, которое есть содержимое элемента управления, либо самому элементу как программному объекту. Этот пример демонстрирует оба варианта ассоциативной связи. Элементу IDC_CHECK1 следует присвоить идентификатор переменной m_check. Нужно также удостовериться, что в раскрывающемся списке Category (Категория) выбрано Value (Значение). Если вы раскроете список Variable type (Тип переменной), то увидите, что вам предоставлен единственный "свободный" выбор — BOOL. Это естественно, поскольку флажок может быть либо установлен, либо сброшен, а значит, ассоциирован только с переменной типа BOOL, которая принимает только два значения — TRUE и FALSE. Щелкните на OK для завершения процедуры.

Ниже перечислены типы переменных, которые могут быть ассоциированы с тем или иным типом элемента управления.

- **Текстовые поля**. Как правило, строковый тип, но иногда и другие — int, float и long.
- **Кнопки**. int.
- **Флажки**. int.
- **Переключатели**. int.
- **Список**. Строковый тип.
- **Поле со списком**. Строковый тип.
- **Полоса прокрутки**. int.

Свяжите таким же образом значение, которое содержится в элементе IDC_EDIT1, с членом-переменной m_edit типа CString, выбрав в раскрывающемся списке Category (Категория) элемент Value (в дальнейшем мы будем говорить о такой связи, как о *связи по значению*). Элемент IDC_LIST1 должен быть связан с членом-переменной m_listbox, который должен быть объектом класса CListBox (в списке Category должно быть выбрано Control). Первый переключатель в группе IDC_RADIO1 должен быть связан с членом-переменной m_radio типа int, причем связь должна быть установлена по значению.

После того как вы щелкнете на OK и тем самым добавите переменную в список членов класса, ClassWizard предложит установить параметры, которые могут быть использованы для проверки достоверности ввода данных. Это делается не для всех видов переменных. Но, например, если речь идет о переменной, связанной с текстовым полем, ClassWizard предлагает

в поле Maximum Characters (Максимум символов) установить максимальную длину вводимой строки (рис. 2.9). Для члена-переменной m_edit установите значение этого параметра равным 10. Если текстовое поле ассоциировано с переменной типа int или float, ClassWizard использует эту же часть окна для установки верхнего и нижнего пределов вводимого пользователем значения. В дальнейшем всю работу по проверке соответствия введенного значения установленным ограничениям и выдачу в случае их нарушения сообщения с просьбой повторить ввод берут на себя функции из библиотеки MFC. Разработчику думать об этих "мелочах" нет никакой нужды.

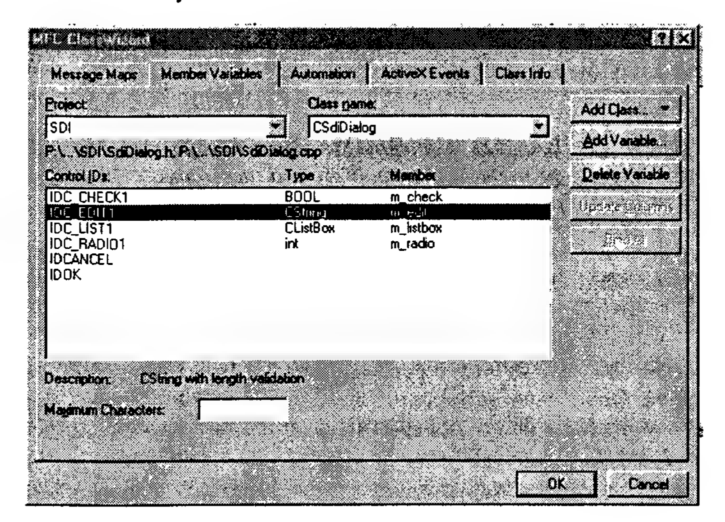


Рис. 2.9. Введите в поле Maximum Characters *число, соответствующее максимальной длине вводимой строки*

Использование класса диалогового окна

Теперь, когда вы сформировали ресурсы окна и подготовили класс окна, можно создавать объект этого класса в самой программе и выводить на экран связанное с ним диалоговое окно. Первый шаг на этом пути — решить для себя, что же именно будет служить "побудительным мотивом" для создания этого окна во время работы программы. Как правило, таким событием является выбор некоторого пункта меню. Но в данном приложении мы не можем использовать событие такого рода по одной простой причине — методы построения меню в приложении будут описаны только в главе 8, *Построение завершенного приложения ShowString*. Так что в нашем примере мы, не мудрствуя лукаво, выведем диалоговое окно на экран сразу же после запуска приложения. В программе для вывода окна на экран нужно вызвать функцию-член класса диалогового окна DoModal().

Немодальные диалоговые окна

Большинство диалоговых окон, которые приходится включать в состав приложения, относятся к так называемым *модальным* окнам. Модальное окно выведено *всегда* поверх всех остальных окон на экране. Пользователь должен поработать в этом окне и обязательно закрыть его прежде, чем приступить к работе в любом другом окне этого же приложения. Примером может служить окно, которое открывается при выборе команды File⇨Open любого приложения Windows.

Немодальное диалоговое окно позволяет пользователю, не закончив работы с ним, "перепрыгнуть" в другое окно приложения, сделать там нечто, затем снова вернуться в немодальное окно и продол-

жить операцию. Типичными немодальными окнами являются окна, которые открываются при отработке команд Edit⇨Find (Правка⇨Поиск) и Edit⇨Replace (Правка⇨Замена) во многих приложениях Windows.

Организация работы с немодальным окном намного сложнее организации работы с модальным. Программировать работу с таким объектом, как диалоговое окно (экземпляром соответствующего класса), нужно очень аккуратно. Как правило, они создаются оператором new, а удаляются — оператором delete при обработке щелчка мышью на кнопке OK или Cancel. Разработчику придется перегрузить большинство методов класса диалогового окна. Короче, прежде чем приступать к программированию приложения с немодальными окнами, нужно достаточно хорошо освоить приемы программирования модальных окон. Когда почувствуете себя готовыми к экспериментам с немодальными окнами, обратите свой взор на пример MODELESS, который имеется в составе программ на Visual C++ в комплекте Visual Studio. Быстрее всего его можно найти, заставив работать систему оперативной справки в режиме поиска MODELESS.

Организация вывода диалогового окна на экран

Выберите вкладку ClassView в рабочей зоне проекта, раскройте пункт SDI Classes, а в нем — CSdiApp. Дважды щелкните на функции-члене InitInstance(). Эта функция вызывается при любом запуске приложения. Перейдите в самое начало файла и после уже имеющихся директив #include вставьте еще одну:

```
#include "sdidialog.h"
```

Теперь при трансляции компилятор будет знать, где взять информацию о классе CSdiDialog.

Перейдите в самый конец текста программы функции и добавьте строки из листинга 2.1 перед оператором return.

Листинг 2.1. Файл SDI.CPP, строки, которые нужно вставить перед окончанием текста функции CSdiApp::InitInstance()

```
CSdiDialog dlg;
dlg.m_check = TRUE;
dlg.m_edit = "hi there";
CString msg;
if (dlg.DoModal() == IDOK)
{
        msg = "You clicked OK. ";
}
else
{
        msg = "You cancelled. ";
}
msg += "Edit Box is: ";
msg += dlg.m_edit;
AfxMessageBox(msg);
```

Вставка фрагмента программного кода

Если вам придется включать новые фрагменты программного кода в заготовки, подготовленные мастером, имеет смысл воспользоваться новыми возможностями, которые предоставляет последняя версия Visual C++. Более детально они будут описаны в приложении B. Функция Autocompletion дает возможность не держать в голове список всех членов класса — переменных и функций. Как только вы введете в текст программы dlg., появится контекстное окно, в котором будут перечислены все члены класса CSdiDialog, включая и унаследованные от базового класса. Если вы начнете вводить идентификатор переменной, например m_, список сместится на переменные-члены, начинающиеся с m_. Теперь можно, пользуясь клавишами управления курсором, выбрать в списке

именно ту, которая вас интересует, и, нажав *клавишу пробела*, вставить ее в текст программы, а затем продолжить ввод. Можно с уверенностью сказать, что этв функция позволит вам значительно ускорить ввод текста программы и избежать ошибок в наборе идентификаторов объектов программы. Правда, иногда раздражает некоторая "неторопливость" функции Autocomplition. В этом случае ничто не мешает вам отключить ее. Для этого придется выбрать команду меню Tools⇨Options (Сервис⇨Параметры), а затвм вкладку Editor (Редактор) и отключить на ней Autocomplition.

Приведенный выше фрагмент программы создает экземпляр класса диалогового окна. Он устанавливает параметры по умолчанию для двух элементов управления — флажка и текстового поля. Программирование других элементов управления — списка и набора переключателей — несколько сложнее, и мы отложим их подробный анализ до разделов *Использование элемента управления типа* список и *Использование элементов управления типа* переключатель этой главы. Сам по себе вывод диалогового окна производится функцией DoModal(), которая возвращает числовое значение — IDOK, если пользователь вышел из окна, щелкнув на OK, и IDCANCEL, если выход произошел после щелчка на Cancel. Затем в приведенном фрагменте формируется сообщение, которое выводится на экран функцией AfxMessageBox().

На заметку Класс CString располагает множеством весьма полезных функций-членов и перегруженных терминальных операторов. Как вы видите в приведенном листинге, терминальный оператор += добавляет символы в конец строки. Более полные сведения о классе CString приведены в приложении E, *Полезные классы*.

Запустите процесс компиляции и компоновки проекта, выбрав команду Build⇨Build или щелкнув на пиктограмме Build (Построить) панели инструментов Build. Запустите выполнение приложения, воспользовавшись командой Build⇨Execute (Построить⇨Выполнить) или щелкнув на пиктограмме Execute (Выполнить) панели инструментов Build. Вы увидите, что на экране появилось диалоговое окно с параметрами элементов управления, установленными по умолчанию в программе, которую вы только что отредактировали. Окно выглядит так, как показано на рис. 2.10. Поменяйте что-либо в текстовом поле, а затем щелкните на OK. После этого на экране должно появиться окно сообщения, в котором будет описано, что вы натворили, как показано на рис. 2.11. Теперь программа готова к дальнейшей работе, но поскольку вы ничего прочего в ней не запрограммировали, не остается ничего другого, как только выбрать File⇨Exit или щелкнуть на кнопке Close в правом углу строки заголовка и закрыть приложение.

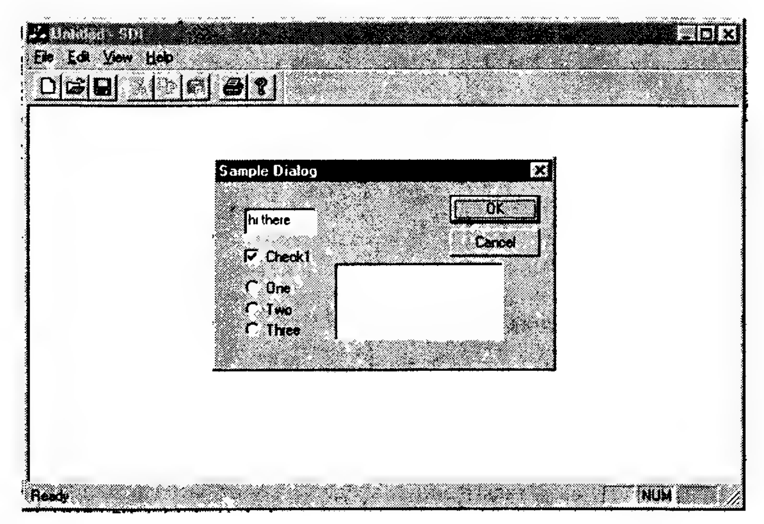


Рис. 2.10. При запуске приложение выведет на экран диалоговое окно

Снова запустите приложение, отредактируйте текст в поле и выйдите из окна, щелкнув на Cancel. Обратите внимание на рис. 2.12, на котором показано окно сообщения, гласящего, что в поле остался исходный текст hi there. Это получилось потому, что MFC не дублирует содержимое текстового поля (как элемента управления) в переменную-член в случае, если пользователь щелкает на Cancel для выхода из окна. И снова, как и в предыдущем эксперименте, закройте приложение.

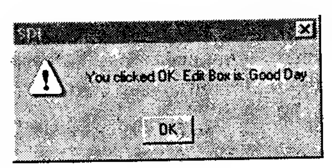


Рис. 2.11. После того как вы щелкнете на OK, приложение выведет в окне сообщения копию текста, введенного в текстовом поле. Надпись в окне гласит: `Вы щелкнули на OK. В текстовом поле: Good Day`

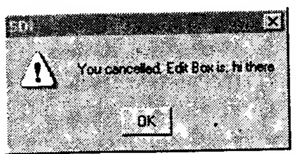


Рис. 2.12. Если вы щелкнете на Cancel, приложение проигнорирует любые изменения элементов управления. Надпись в окне гласит: `Вы отказались. В текстовом поле: hi there`

А теперь посмотрите, как будет реагировать приложение на попытку ввести более 10 символов в текстовом поле (при настройке ресурсов с помощью ClassWizard для этого параметра было выбрано ограничение 10). Вы увидите, что все символы после десятого игнорируются, при этом еще и раздается мелодичный (или противный — это как кто воспринимает) писк. Если вы попытаетесь вставить в поле текст из системного буфера, то будут восприняты только первые 10 символов.

За кулисами

Сейчас вы гадаете, как же это все там происходит? Когда вы щелкнули на OK в диалоговом окне, MFC организовал вызов функции OnOK(). Эта функция унаследована от базового класса CDialog, классом-наследником которого является CSdiDialog. Помимо прочего, в ней вызывается DoDataExchange(), подготовленная средствами ClassWizard. Вот как она выглядит в настоящий момент.

```
void CSdiDialog::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CSdiDialog)
        DDX_Control(pDX, IDC_LIST1, m_listbox);
        DDX_Check(pDX, IDC_CHECK1, m_check);
        DDX_Text(pDX, IDC_EDIT1, m_edit);
        DDV_MaxChars(pDX, m_edit, 10);
        DDX_Radio(pDX, IDC_RADIO1, m_radio);
        //}}AFX_DATA_MAP
}
```

Все функции, имена которых начинаются с DDX, выполняют обмен данными. Вторым аргументом каждой функции является идентификатор элемента управления, а третьим — идентификатор члена-переменной класса. Именно таким образом ClassWizard устанавливает соответствие между элементами управления и членами класса диалогового окна — это ClassWizard подготовил такой текст программы вместо вас. Вспомните, что ClassWizard также включил имена членов-переменных в файл заголовка, в котором объявляется структура класса.

Имеется 34 функции; их имена начинаются с DDX — одна на каждый тип данных, которыми могут обмениваться диалоговое окно и соответствующий класс. Каждая функция включает в свое имя также имя элемента управления. Например, функция DDX_Check() используется

для связи между элементом типа флажок (check box) и членом-переменной типа BOOL. Аналогично DDX_Text() используется для связи члена-переменной типа CString с текстовым полем. ClassWizard выбирает соответствующую функцию в процессе выполнения описанной выше операции связывания членов класса с элементами управления.

На заметку Существует несколько DDX-функций, за которые ClassWizard не несет ответственности. Например, если вы связываете список по значению с переменной, то единственным выбором для вас является тип CString. В этом случае ClassWizard формирует функцию DDX_LBString(), которая связывает выбранный в списке элемент с членом-переменной типа CString. Однако иногда эффективнее использовать индекс выбранного элемента, а не сам элемент. Для этого случая имеется функция DDX_LBIndex(), которая выполняет соответствующий обмен. Вызов этой функции можно добавить в текст функции-члена DoDataExchange(). Соответствующая строка может быть вставлена в том месте программы, где имеется специальный комментарий, созданный ClassWizard. При этом не забудьте добавить соответствующую переменную-член в объявление класса в файле заголовка. Полный список DDX-функций можно найти в электронной документации.

Функции, имена которых начинаются с DDV, ответственны за проверку соблюдения заданных ограничений на вводимые данные. ClassWizard вставляет вызов DDV_MaxChars() сразу за вызовом DDX_Text, которая передает содержимое текстового поля IDC_EDIT1 в переменную m_edit. Первый аргумент вызова функции — идентификатор переменной-члена, а второй — значение параметра, ограничивающего длину вводимой строки. Если пользователь когда-нибудь при работе с программой попытается ввести символов больше, чем дозволено, то на этот случай в тексте DDV_MaxChars()есть специальный фрагмент, который организует вывод на экран предупреждающего сообщения с приглашением повторить попытку. Так что вы можете только заказать величину ограннчения и рассчитывать, что все дальнейшее будет оргаиизовано ClassWizard и MFC наилучшим образом.

Использование элемента управления типа список

Работать со списками значительно сложнее, поскольку реальным объектом список становится только тогда, когда диалоговое окно уже выведено на экран. Нельзя вызывать функции-члены списка до тех пор, пока не будет порожден экземпляр класса диалогового окна. То же самое справедливо и для всех прочих элементов управления, которые входят в его состав. Таким образом, инициализировать список — заполнить его элементамн-строкамн — и анализировать, какая строка выбрана, можно только иа том участке программы, который выполняется во время присутствия диалогового окна на экране.

Когда наступает время инициализировать диалоговое окно, перед выводом его на экран, вызывается функция-член класса CDialog OnInitDialog(). Поскольку полного описания всего, что необходимо сделать для этого, придется подождать до главы 3, сейчас вы должны поверить автору на слово и следовать предлагаемой ннже последовательности действий, целью которых является включение этой функции в состав членов класса.

В окне ClassView щелкните правой кнопкой мыши на CSdiDialog и выберите в контекстном меню команду Add Windows Message Handler (Добавить обработчик сообщений Windows). На экране появится окно New Windows Message and Event Handlers (Новые обработчики сообщений Windows и событий), показанное на рис. 2.13. В списке New Windows messages/events (Новые сообщения Windows/события) выберите WM_INITDIALOG и щелкните на кнопке Add Handler (Добавить обработчик). Идентификатор сообщения исчезнет из списка в левой части окна и появится в списке Existing message/event handlers (Существующие обработчики сообщений/событий) в правой его части. Щелкните на нем и затем щелкните на кнопке Edit Existing (Правка существующих) с тем, чтобы увидеть текст программы.

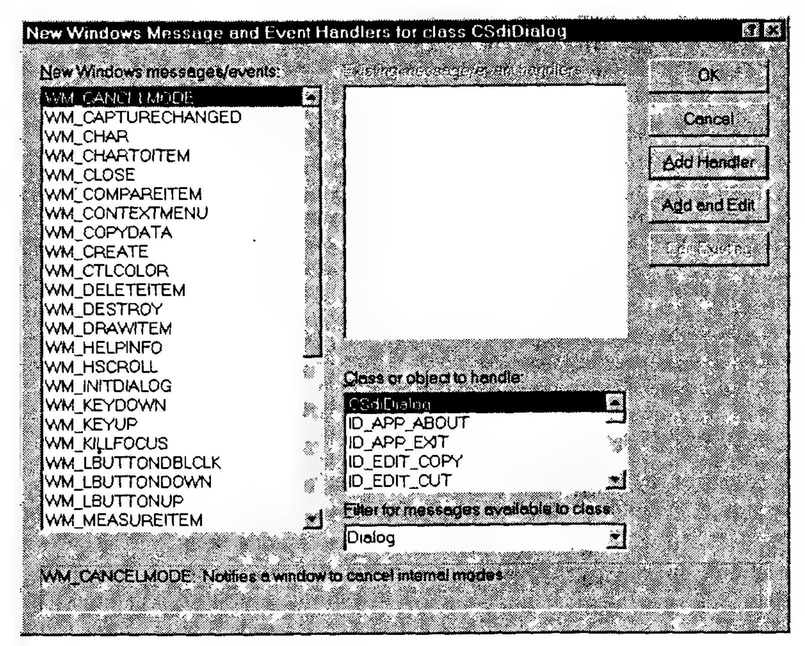


Рис. 2.13. Диалоговое окно New Windows Message and Event Handlers *поможет перегрузить функцию* OnInitDialog()

Удалите комментарий TODO (СДЕЛАТЬ) и вставьте вызовы функций-членов списка. В результате текст программы должен выглядеть так, как в листинге 2.2.

Листинг 2.2. Файл SDIDIALOG.CPP — CSdiDialog::OnInitDialog()

```
BOOL CSdiDialog::OnInitDialog()
{
        CDialog::OnInitDialog();

        m_listbox.AddString("First String");
        m_listbox.AddString("Second String");
        m_listbox.AddString("Yet Another String");
        m_listbox.AddString("String Number Four");
        m_listbox.SetCurSel(2);

        m_radio = 1;
        UpdateData(FALSE);

        return TRUE;  // Возвращает TRUE, если только вы не установили фокус
                      //  ввода на элемент управления.
                      // ИСКЛЮЧЕНИЕ: страницы свойств OCX должны возвращать FALSE.
```

Работа этого фрагмента программы начинается с вызова той версии функции OnInitDialog(), которая принадлежит базовому классу CDialog. Она выполнит все фоновые операции, предусмотренные в MFC для инициализации любого диалогового окна. Затем вызывается функция-член класса объектов список AddString(), которая, как вы, очевидно, догадались по ее названию, добавляет в список новый элемент — строку. Пользователь увидит на экране элементы в том порядке, в котором они включались в список в программе. После нескольких вызовов AddString() расположен вызов функции SetCurSel(), которая должна установить начальный фокус выбора в списке. Индекс, который передается в SetCurSel(), на-

чинается с 0. Поэтому, когда программа будет выполняться, выбранным по умолчанию на экране будет третий элемент списка, который имеет индекс 2.

На заметку Как правило, элементы в списке не устанавливаются так жестко, как в приведенном выше примере. Для того чтобы иметь возможность заполнять список программно, нужно добавить в состав класса диалогового окна член-переменную — экземпляр класса CStringArray и функцию-член, которая добавляла бы строки в этот массив. Тогда функция OnInitDialog() сможет использовать этот массив, чтобы заполнить список. Альтернативный вариант — использовать какой-либо другой класс из набора, которым располагает MFC, или даже заполнить список из базы данных. Подробнее о CStringArray и других классах MFC можно узнать из приложения E, а о базах данных — из главы 22, *Доступ к базам данных*.

Для того чтобы организовать вывод в окне сообщения информации о выбранном из списка элементе, нужно также включить специальный член-переменную в класс диалогового окна. В эту переменную при закрытии диалогового окна будет записываться значение, к которому затем можно обратиться, несмотря на то что окно уже закрыто. В окне ClassView щелкните правой кнопкой мыши на CSdiDialog и выберите в контекстном меню команду Add Member Variable (Добавить член-переменную). Заполните реквизиты в появившемся диалоговом окне, как это показано на рис. 2.14, и щелкните на OK. Тем самым вы включите в состав класса член-переменную m_selected типа CString, что и будет зафиксировано в файле заголовка класса диалогового окна. Если планируется, что список должен поддерживать многозначный выбор, то тип введенной переменной нужно указать как CStringArray, т.е. эта переменная будет содержать массив выбранных пользователем элементов списка. Строго говоря, этот член должен быть закрытым и нужно либо добавить открытую функцию-член доступа к этому члену-переменной, либо объявить CSdiApp::InitInstance() *дружественной* функцией класса CSdiDialog с тем, чтобы обеспечить полное соответствие концепциям объектно-ориентированного программирования. Сейчас мы рассматриваем простенький пример, в котором эти требования не соблюдаются, но в "настоящем" проекте следует к этому относиться серьезно — член-переменная должен быть закрытым (private).

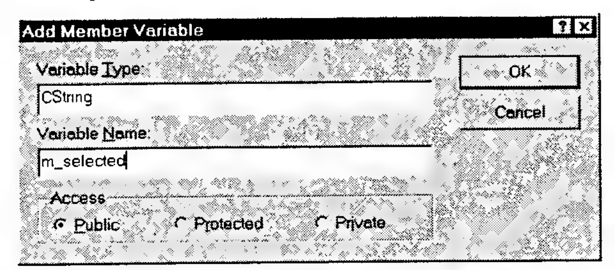


Рис. 2.14. Включите в состав класса переменную типа CString, которая будет хранить выбранный пользователем элемент списка



Краткий обзор основных концепций объектно-ориентированного программирования вы найдете в приложении A. В частности, там вы сможете освежить свои знания о функциях доступа к членам класса и дружественных функциях класса.

Эта новая переменная будет хранить выбранный пользователем элемент списка — строку. Запись в нее производится тогда, когда пользователь щелкает на OK или Cancel. Для того чтобы добавить функцию, которая будет вызываться после щелчка на OK, выполните следующее.

1. В окне ClassView щелкните правой кнопкой мыши на CSdiDialog и выберите в контекстном меню Add Windows Message Handler.
2. В окне New Windows Message and Event Handlers, показанном на рис. 2.15, выделите в списке Class or object to handle (Классы или объекты для обработки) IDOK.
3. В левом списке, New Windows messages/events, выберите BN_CLICKED. Этим вы добавляете функцию для обработки однократного щелчка на кнопке OK.
4. Щелкните на кнопке Add Handler. Появится диалоговое окно Add Member Function (Добавить функцию-член), которое показано на рис. 2.16.
5. Согласитесь с предложенным именем функции OnOK(), для чего щелкните на OK.
6. Щелкните на кнопке Edit Existing, чтобы увидеть текст программы. Добавьте в него несколько строк так, чтобы он приобрел вид, показанный в листинге 2.3.

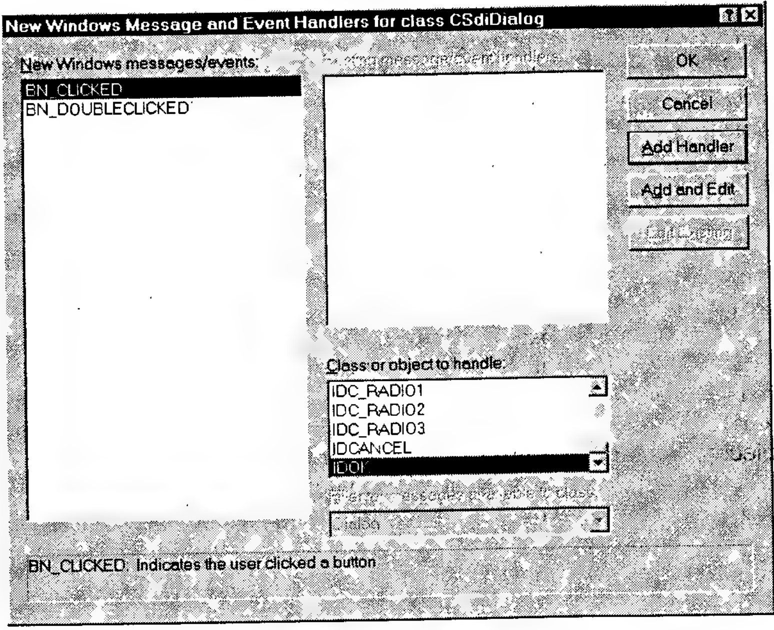


Рис. 2.15. Добавление функции обработки щелчка на OK в новом диалоговом окне

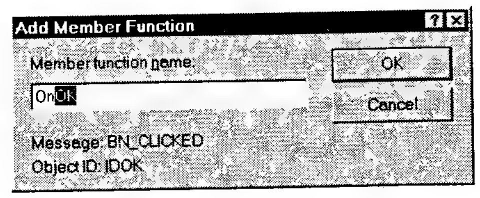


Рис. 2.16. ClassWizard предлагает очень удачное имя для этой функции обработки события. Не меняйте его

Листинг 2.3. Файл SDIDIALOG.CPP — CSdiDialog::OnOK()

```
void CSdiDialog::OnOK()
{
        int index = m_listbox.GetCurSel();
        if (index != LB_ERR)
        {
                m_listbox.GetText(index, m_selected);
        }
        else
        {
                m_selected = "";
        }

        CDialog::OnOK();
}
```

Работа этого фрагмента программы начинается с вызова функции-члена класса объектов список GetCurSel(), которая возвращает константу LD_ERR в случае, если не выбран ни один элемент списка или если выбрано более одного элемента. Иначе возвращается индекс выбранного элемента (еще раз напомним, что нижняя граница индекса — 0). Функция-член того же класса GetText() переписывает строку выбранного элемента в переменную-член m_selected класса CSdiDialog. Первым аргументом функции является индекс выбранного элемента. После этого вызывается функция-член OnOK()базового класса CDialog, которая выполняет все стандартные действия по закрытию окна.

Сейчас можно заодно и скорректировать текст функции CSdiApp::InitInstance()с тем, чтобы в выведенном после закрытия окна сообщении было упомянуто, какой выбор сделал пользователь в списке. Эти строки программы будут выполняться независимо от того, каким образом было закрыто окно — щелкнул пользователь на OK или на Cancel. Сначала нужно создать дополнительно функцию, которая обрабатывала бы щелчок на Cancel. Такая функция — OnCancel() — создается точно таким же образом, как и OnOK(), но в правом списке, Class or object to handle, выделите IDCANCEL и согласитесь с именем функции OnCancel(). Как видно из листинга 2.4 сформированной функции, переменная m_selected очищается, поскольку пользователь отказался от диалога с программой.

Листинг 2.4. Файл SDIDIALOG.CPP — CSdiDialog::OnCancel()

```
void CSdiDialog::OnCancel()
{
        m_selected = "";
        CDialog::OnCancel();
}
```

В функцию CSdiApp::InitInstance()добавьте следующие строки перед обращением к функции AfxMessageBox():

```
msg += ". List Selection: ";
msg += dlg.m_selected;
```

Оттранслируйте, запустите приложение и поэкспериментируйте с ним. Работает ли оно, как ожидалось? Получили ли вы на экране картинку, сходную с приведенной на рис. 2.17?

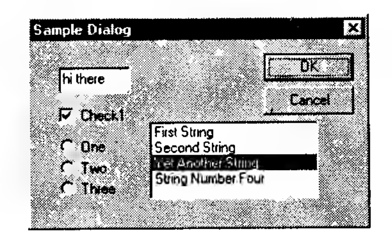


Рис. 2.17. Теперь приложение выводит элементы списка

Использование элементов управления типа переключатель

Вы уже, наверное, обратили внимание на то, что при выводе диалогового окна на экран ни один из переключателей в группе не выбран. Можно так организовать программу, что в этой ситуации один из них будет выбран по умолчанию. Для этого требуется просто добавить пару строк в текст функции CSdiDialog::OnInitDialog(). Следующий фрагмент программы включит второй переключатель и передаст изменения в диалоговое окно:

```
m_radio = 1;

UpdateData(FALSE);
```

Вам придется вспомнить, что m_radio является членом-переменной, с которым связана группа переключателей. Она (переменная) представляет собой индекс выбранного переключателя в этой группе элементов управления (как всегда, индекс начинается с 0). Так что значение индекса 1 соответствует второму переключателю в группе. Вызов функции UpdateData()в этом фрагменте обновляет содержимое элементов управления диалогового окна соответственно состоянию связанных с ними переменных-членов. Аргумент функции UpdateData()указывает направление передачи данных: UpdateData(TRUE) обновило бы содержимое переменных соответственно элементам управления, т.е. переписало бы в m_radio индекс выбранного в группе переключателя.

В отличие от списка, группа переключателей доступна и после того, как диалоговое окно убрано с экрана. Так что вам не придется добавлять что-либо в функции OnOK()и OnCancel(). Вместо этого у вас будет другая проблема — как преобразовать целое значение индекса в строковое выражение, которое нужно будет добавить в "хвост" текста сообщения в msg. Существует множество ее решений, включая функцию-член Format() класса CString, но в данном случае можно поступить гораздо проще — использовать оператор switch, поскольку индекс может принимать лишь ограниченное множество значений. В конец текста функции CSdiApp::InitInstance(), перед вызовом AfxMessageBox(), добавьте несколько строк, представленных в листинге 2.5.

Листинг 2.5. Файл SDIDIALOG.CPP — строки, которые нужно включить в CSdiApp::InitInstance()

```
msg += "\r\n";
msg += "Radio Selection: ";

switch (dlg.m_radio)
{
case 0:
        msg += "0";
        break;
case 1:
        msg += "1";
        break;
case 2:
        msg += "2";
        break;
default:
        msg += "none";
        break;
}
```

Первая из новых строк добавляет в сообщение два специальных символа — *перевод каретки* \r и *перевод строки* \n, — которые в совокупности представляют маркер конца строки для Windows. В результате дальнейшая часть сообщения msg начнется с новой строки. Оператор switch не представляет ничего нового для тех, кто знаком с языком C++ или хотя бы с языком C. Он организует выполнение одного из операторов, следующих за case, в зависимости от значения dlg.m_radio.

Теперь в очередной раз оттранслируйте и запустите приложение. Никаких сюрпризов? То, что появилось на экране, должно быть очень похоже на представленное на рис. 2.18. Вам придется формировать и использовать диалоговые окна несчетное число раз, пока вы будете выполнять упражнения из этой книги. Так что не пожалейте времени на то, чтобы разобраться до мелочей в процессе формирования подобного рода объектов. Можно пройтись по всем стадиям выполнения программы с помощью отладчика, просматривая промежуточные значения переменных. О технологии использования отладчика можно справиться в приложении Г, *Отладка*. Частично об отладке будет рассказано и в главе 24.

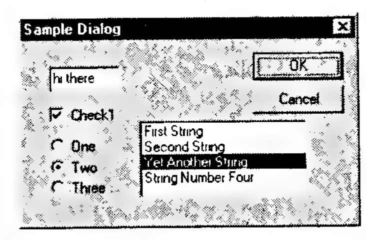


Рис. 2.18. Теперь в приложении по умолчанию выбран переключатель Two

ГЛАВА 3

Сообщения и команды

В этой главе...

Обработка сообщений

Циклы обработки сообщений

Карты сообщений

Как мастер ClassWizard помогает перехватывать сообщения

Список сообщений

Команды

Обновление команд

Как ClassWizard помогает перехватывать команды и их обновления

Обработка сообщений

Если и существует некоторая особенность, отличающая программирование в Windows от других областей программирования, то это сообщения. Большинство DOS-программ, например, основано на отслеживании возможных источников поступления информации, таких как клавиатура и мышь, ожидая ввода от них. Программа, которая не следит за мышью, не сможет реагировать на поступающие от нее сигналы. Что же касается Windows-программ, то там все происходит с точностью до наоборот: программа *управляется* сообщениями. Сообщения являются тем средством, с помощью которого операционная система может дать знать приложению, что что-то произошло, например пользователь нажал клавишу на клавиатуре или щелкнул кнопкой мыши, или передвинул мышь, или подготовил принтер к выводу информации. Окно, а каждый информационный элемент на экране есть своего рода окно, также может посылать сообщения другому окну и, как правило, большинство окон реагирует на полученное сообщение тем, что пересылает его дальше, третьему окну, слегка видоизменив. Значительную помощь в организации работы с сообщениями оказывает MFC, скрывая от программиста многие подробности процесса, но грамотный разработчик всегда должен представлять себе, что же происходит там, под ковром.

Хотя операционная система и использует целые числа для идентификации событий, в тексте программы мы будем иметь дело с символьными идентификаторами. Огромное количество директив `#define` связывает символьные идентификаторы с соответствующими числами и позволяет программистам в разговоре между собой в кругу посвященных манипулировать словечками вроде `WM_PAINT` и `WM_SIZE`. Префикс `WM` означает *Window Message* (сообщение Windows). Фрагмент перечня сообщений представлен в листинге 3.1.

Листинг 3.1. Фрагмент файла `winuser.h` — определение кодов сообщений

```
#define WM_SETFOCUS          0x0007
#define WM_KILLFOCUS         0x0008
#define WM_ENABLE            0x000A
#define WM_SETREDRAW         0x000B
#define WM_SETTEXT           0x000C
#define WM_GETTEXT           0x000D
#define WM_GETTEXTLENGTH     0x000E
#define WM_PAINT             0x000F
#define WM_CLOSE             0x0010
#define WM_QUERYENDSESSION   0x0011
#define WM_QUIT              0x0012
#define WM_QUERYOPEN         0x0013
#define WM_ERASEBKGND        0x0014
#define WM_SYSCOLORCHANGE    0x0015
#define WM_ENDSESSION        0x0016
```

Сообщению известно, для какого окна оно предназначено. Оно может иметь до двух параметров. Часто в эти два параметра упаковывается несколько совершенно различных величин, но это уже другое дело.

Обработка разных сообщений выполняется разными компонентами операционной системы и приложения. Например, когда пользователь передвигает мышь по полю окна, формируется сообщение `WM_MOUSEMOVE`, которое передается окну, а окно, в свою очередь, передает это сообщение операционной системе. И уже последняя перерисовывает указатель мыши в новом месте. Когда пользователь щелкает левой кнопкой мыши на экранной кнопке, кнопка, которая также есть особый вид окна, получает сообщение `WM_LBUTTONDOWN`. В процессе обработки этого сообщения кнопка часто формирует новое сообщение для окна, в котором она находится, причем это сообщение гласит: “Ой, на мне щелкнули!”.

Библиотека MFC позволяет программистам в подавляющем большинстве случаев полностью отстраниться от сообщений нижнего уровня, таких как WM_MOUSEMOVE и WM_LBUTTONDOWN. Программист может полностью сосредоточиться на сообщениях более высокого уровня, которые гласят что-нибудь вроде "Выбран третий элемент такого-то списка" или "Произошел щелчок на кнопке **Move**". Поступают такого рода сообщения в те программы, которые пишет программист, и в компоненты операционной системы точно так же, как и сообщения нижнего уровня. Единственная разница в том, что MFC берет на себя значительную часть работы по обработке сообщений низкого уровня и позволяет заметно облегчить распределение сообщений между разными классами объектов, на уровне которых и будет производиться их обработка. В программах на языке С, которые были созданы по старой технологии, такое объявление выполнялось на достаточно высоком уровне взаимодействия между собственно языком и системой Windows. При этом в дело вступали различные средства объектно-ориентированного программирования, которые позволяли в максимальной степени скрывать детали выполняемых операций внутри объектов.

Циклы обработки сообщений

Сердцем любой Windows-программы является *цикл обработки сообщений* (Message Loop), который практически всегда находится в функции WinMain(). Эта функция в Windows-приложениях играет ту же роль, что и функция Main()в DOS-приложениях, — ее вызывает операционная система сразу же после загрузки приложения в память. К большому облегчению программистов, теперь они могут не отвлекаться на набивку текста WinMain(), поскольку это сделает AppWizard. Но это не значит, что сама функция исчезла. Текст типичной функции WinMain()представлен в листинге 3.2.

Листинг 3.2. Типичная функция WinMain()

```
int APIENTRY WinMain(HINSTANCE hInstance,
          HINSTANCE hPrevInstance,
          LPSTR lpCmdLine,
          int nCmdShow)
{
   MSG   msg;

   if( !InitApplication( hInstance))
      return (FALSE);

   if( !InitInstance( hInstance, nCmdShow))
      return (FALSE);

   while( GetMessage( &msg, NULL, 0, 0)) {
      TranslateMessage( &msg);
      DispatchMessage( &msg);
   }
   return (msg.wParam);
}
```

В С-программах для Windows, похожих на эту, функция InitApplication() вызывает RegisterWindow(), а InitInstance() — CreateWindow(). Детали этого процесса вы найдете в приложении Б. Затем наступает очередь цикла обработки сообщений. Он представляет собой типичную циклическую конструкцию С на базе оператора while, внутри которой вызывается функция GetMessage(). Эта функция API заполняет msg кодом сообщения, которое операционная система распределила для этого приложения, и почти всегда возвращает TRUE. Таким

образом, цикл повторяется снова и снова до тех пор, пока работает приложение. Единственный вариант, при котором GetMessage()возвращает FALSE, — получение сообщения WM_QUIT.

При работе с сообщениями, поступающими с клавиатуры, некоторую часть предварительной обработки берет на себя функция API TranslateMessage(). Ее назначение состоит в следующем. Прикладной части программы нет дела до сообщений наподобие "Нажата клавиша <A>" и "Отпущена клавиша <A>". Прикладную часть, в конце концов, интересует только то, какую литеру (символ) ввел пользователь, т.е. ее вполне удовлетворит сообщение "Введен символ А". Вот это преобразование — нескольких сообщений о деталях процесса в одно сообщение о его сути — и выполняет функция TranslateMessage(). Она перехватывает сообщения WM_KEYDOWN и WM_KEYUP и вместо них посылает сообщение WM_CHAR. Конечно, если пользоваться библиотекой MFC, то такие мелочи, как ввод символа **А**, проходят, как правило, мимо вас. Пользователь вводит текст в текстовое поле или в другой элемент управления, и забота программиста — извлечь введенный текст из этого объекта после того, как пользователь щелкнет на **OK**. Как был организован прием символов с клавиатуры, теперь уже не наше дело. Таким образом, на функцию TranslateMessage() можно не обращать особого внимания.

Функция API DispatchMessage() вызывает, в свою очередь, функцию WndProc() того окна, для которого предназначено сообщение. Типичная функция WndProc() в С-программе для Windows представляет собой огромный оператор switch с отдельными case для каждого сообщения, которое приложение намеревается самостоятельно обрабатывать. Текст ее приведен в листинге 3.3.

Листинг 3.3. Типичная функция WndProc()

```
LONG APIENTRY MainWndProc( HWND hwnd,    // Дескриптор окна.
       UINT msg,        //Тип сообщения.
       UINT wParam,     // Дополнительная информация.
       LONG lParam)     // Дополнительная информация.
{

   switch( msg){
      case WM_MOUSEMOVE: {
          // Обработка перемещения мыши.
         break;

      case WM_LBUTTONDOWN: {
          // Обработка щелчка левой кнопкой мыши.
         break;

      case WM_RBUTTONDOWN: {
          // Обработка щелчка правой кнопкой мыши.
         break;

      case WM_PAINT :
          // Перерисовать окно.
         break;

      case WM_DESTROY :     // Сообщение: окно будет уничтожено.
         PostQuitMessage( 0);
         return 0;
         break;

      default:
         return (DefWindowProc( hwnd, msg, wParam, lParam));
   }
   return (0);
}
```

Вы, конечно, можете себе представить, какой длины достигает подобная функция в более или менее порядочном приложении. Сопровождение такой программы программистом зачастую становится причиной ночных кошмаров. MFC решает проблему следующим образом — информация о сообщениях, которые должны обрабатываться, расположена поближе к функциям, которые и должны выполнять обработку. Таким образом, отпадает необходимость в огромных операторах switch, в которых сосредоточено распределение сообщений. Читайте дальше — и узнаете, как это делается.

Карты сообщений

Использование *карты сообщений* (Message maps) лежит в основе подхода, который реализуется в MFC для программирования Windows-приложений. Суть его состоит в том, что от разработчика требуется только написать функции обработки сообщений и включить в свой класс карту сообщений, которая фактически скажет: "Я буду обрабатывать такое-то сообщение". После этого главная программа будет отвечать за то, чтобы сообщение было передано именно той функции, которая будет его обрабатывать. В результате исключается необходимость разрабатывать функцию WinMain(), которая должна была бы передавать сообщения другой функции — WndProc() (ее также нужно было бы разработать), а последняя должна была бы анализировать сообщение и вызывать соответствующую функцию обработки.



Если вам приходилось работать с Visual Basic, то вы наверняка знакомы с такими процедурами обработки событий (event procedure), как, например, "щелчок мышью". Функция обработки сообщения в программе на C++ играет ту же роль, что и процедура обработки события. Карта сообщений — это способ связать событие с его обработчиком.

Карта сообщений состоит из двух частей: одна — в файле заголовка для класса .h, а другая — в соответствующем файле реализации .cpp. Они, как правило, формируются мастерами, хотя в некоторых случаях вы можете сделать это (или частично отредактировать их) и самостоятельно. В листинге 3.4 представлена часть текста файла заголовка одного из классов простого приложения ShowString. Само приложение будет рассмотрено в главе 8.

Листинг 3.4. Карта сообщений из файла ShowString.h

```
//{{AFX_MSG(CShowStringApp)
afx_msg void OnAppAbout();
   // ВНИМАНИЕ!!  Здесь ClassWizard будет добавлять и
   //    удалять функции-члены.
   //    НЕ РЕДАКТИРУЙТЕ текст в этих блоках!
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
```

Здесь объявляется функция OnAppAbout(). Специальным образом оформленный комментарий позволяет ClassWizard определить, какие именно сообщения перехватываются этим классом. DECLARE_MESSAGE_MAP — это макрос, расширяемый препроцессором компилятора Visual C++, в котором объявляются переменные и функции, принимающие участие в этом фокусе с перехватом сообщений.

Карта сообщений в файле .cpp, как показано в листинге 3.5, также достаточно проста.

Листинг 3.5. Карта сообщений из файла ShowString.cpp

```
BEGIN_MESSAGE_MAP(CShowStringApp, CWinApp)
   //{{AFX_MSG_MAP(CShowStringApp)
   ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
   //}}AFX_MSG_MAP
   // Стандартные команды для файловых документов.
   ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
   ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
   // Стандартные команды настройки принтера.
   ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()
```

Макросы карты сообщений

Макросы BEGIN_MESSAGE_MAP и END_MESSAGE_MAP, так же как DECLARE_MESSAGE_MAP в файле заголовка, объявляют члены (переменные и функции), которые программа должна использовать для того, чтобы разобраться в картах всех объектов системы. Существует довольно большой набор макросов, используемых для работы с картой сообщений. Некоторые из них перечислены ниже.

- DECLARE_MESSAGE_MAP. Используется в файле заголовка для того, чтобы объявить, что в файл собственно текста программы будет включена карта сообщений.
- BEGIN_MESSAGE_MAP. Отмечает начало карты сообщений в тексте программы.
- END_MESSAGE_MAP. Отмечает конец карты сообщений в тексте программы.
- ON_COMMAND. Используется для того, чтобы перенаправить обработку некоторой команды функции-члену класса.
- ON_COMMAND_RANGE. Используется для того, чтобы перенаправить обработку группы команд, идентификаторы которых лежат в заданном диапазоне, одной функции-члену класса.
- ON_CONTROL. Используется для того, чтобы перенаправить обработку кода извещения от элемента управления, введенного программистом, функции-члену класса.
- ON_CONTROL_RANGE. Используется для того, чтобы перенаправить обработку группы кодов извещений, значения которых находятся в заданном диапазоне, одной функции-члену класса.
- ON_MESSAGE. Используется для того, чтобы перенаправить обработку некоторого сообщения, введенного программистом, функции-члену класса.
- ON_REGISTERED_MESSAGE. Используется для того, чтобы перенаправить обработку некоторого зарегистрированного сообщения, введенного программистом, функции-члену класса.
- ON_UPDATE_COMMAND_UI. Используется для того, чтобы перенаправить обновление, связанное с заданной командой, функции-члену класса.
- ON_COMMAND_UPDATE_UI_RANGE. Используется для того, чтобы перенаправить одной и той же функции-члену класса обновление, связанное с группой команд, идентификаторы которых находятся в заданном диапазоне значений.
- ON_NOTIFY. Используется для того, чтобы перенаправить функции-члену класса обработку заданного кода извещения, который сопровождается дополнительными данными от элемента управления.

- ON_NOTIFY_RANGE. Используется аналогично макросу ON_NOTIFY, но применяется для группы элементов, извещения от которых должны обрабатываться одной и той же функцией-членом класса. Группа специфицируется интервалом кодов идентификаторов элементов управления, которые являются дочерними окнами по отношению к тому окну, которое эти сообщения перехватывает.
- ON_NOTIFY_EX. Используется для того, чтобы перенаправить обработку заданного кода извещения, сопровождаемого дополнительными данными от элемента управления, функции-члену класса. Последняя, в свою очередь, должна вернуть TRUE или FALSE с тем, чтобы сигнализировать, нужно ли передавать дальше это уточненное сообщение другому объекту для возможной реакции.
- ON_NOTIFY_EX_RANGE. Используется аналогично макросу ON_NOTIFY_EX, но применяется для группы элементов, извещения от которых должны обрабатываться одной и той же функцией-членом класса. Элементы управления, которые передают сообщения такого формата, являются дочерними окнами по отношению к тому окну, которое эти сообщения перехватывает.

В дополнение к перечисленным существует еще около ста макросов, по одному на каждое стандартное сообщение, которые направляют соответствующее сообщение функции-члену. Например, ON_CREATE направляет сообщение WM_CREATE функции On_Create(). При пользовании такими макросами нельзя изменять имена функций. Эти макросы, как правило, включаются в карту сообщений самим ClassWizard, что будет продемонстрировано в главе 8.

Что происходит с картой сообщений

Карты сообщений, приведенные в листингах 3.4 и 3.5, созданы для класса CShowStringApp приложения ShowString. Этот класс отвечает за выполнение задач достаточно высокого уровня, например открытие нового файла или отображение окна About. Компоненты, которые добавлены в карту сообщенний файла заголовка, могут быть истолкованы следующим образом: "Существует функция OnAppAbout(), которая не имеет параметров". Компонент, который добавлен в собственно текст программы (файл .cpp), означает: "Когда придет сообщение от команды ID_APP_ABOUT, вызовите OnAppAbout()". Ничего неожиданного в том, что функция-член OnAppAbout()выведет на экран окно About этого приложения, естественно, нет.

Но что же в действительности происходит при этом с картой сообщений? Каждое приложение имеет объект, который является наследником класса CWinApp, и имеет функцию-член Run(). Эта функция обращается к CWinThread::Run(), которая значительно длиннее, чем продемонстрированная вам ранее функция WinMain(), но имеет точно такой же цикл обработки сообщений — вызов GetMessage(), вызов TranslateMessage()и вызов DispatchMessage(). Почти все объекты-окна используют тот же самый класс окна, характерный для прежней технологии программирования, и ту же самую функцию WndProc(), но теперь названную AfxWndProc(). Функция WndProc(), как вы уже видели, знает дескриптор окна hWnd, для которого предназначено сообщение. Библиотека MFC, в свою очередь, содержит нечто, называемое *картой дескрипторов* (handle map), — таблицу дескрипторов окон и указателей объектов. Таким образом, главная программа может, используя всю эту информацию, найти указатель на объект cWnd *. Далее она вызывает WindowProc() — виртуальную функцию этого объекта. Кнопки или окна представления, естественно, имеют разные реализации этой функции, но волшебные свойства полиморфизма приводят к тому, что вызывается именно та реализация, которая нужна.

Полиморфизм

Виртуальные функции и *полиморфизм* — это базовые понятия концепции объектно-ориентированного программирования. Они должны быть известны любому программисту, работающему на языке C++, особенно если он пользуется библиотеками такого класса, как MFC. С ними встречаешься всякий раз при использовании указателей на объекты, особенно если эти объекты принадлежат классам, производным от других классов. В качестве примера рассмотрим некоторый класс CDerived, производный от класса CBase. Пусть этот класс имеет функцию-член Function(), которая объявлена в базовом классе и перегружена в производном классе. Теперь в нашем распоряжении две функции: полное имя одной — CBase::Function(), а другой — CDerived::Function().

Если в тексте вашей программы объявлен указатель на объект базового класса, а вы присваиваете ему адрес объекта производного класса, то обращение к функции может иметь следующий вид:

```
CDerived derivedobject;
CBase* basepointer;
basepointer = &derivedobject;
basepointer->Function();
```

В этом случае будет вызвана функция CBase::Function(). Но может оказвться, что это совсем не то, что вам нужно. Хотя и используется указатель на объект класса CBase, в действительности вам нужно вызвать функцию класса CDerived. Чтобы указать на такой способ обращения к функции Function(), она объявляется в базовом классе как *виртуельная функция*. Это можно интерпретировать, как указание компилятору перегрузить данную функцию при первой же возможности в производном классе.

Поскольку функция Function()объявлена в базовом классе CBase как виртуальная, приведенный выше фрагмент программы действительно вызовет CDerived::Function(), как того и добивался разработчик. Это и есть полиморфизм на практике. Подобные случаи вы сможете неоднократно встретить при работе с классами, объявленными в MFC. Например, вы используете для обращения к объекту производного класса CButton или CView (или любого аналогичного) указатель, объявленный как CWnd*. Затем, если вызывается функция типа WindowProc(), компилятор сформирует обращение к функции-члену производного класса CButton::WindowProc().

Функция WindowProc() вызывает OnWndMsg() — функцию C++, которая собственно и обрабатывает сообщения. Во-первых, она проверяет, что же это было — сообщение, команда или код извещения. Предположим, поступило сообщение. Тогда функция просматривает карту сообщений для своего класса, используя члены класса (переменные и функции), которые были установлены макросами BEGIN_MESSAGE_MAP, END_MESSAGE_MAP и DECLARE_MESSAGE_MAP. Помимо всего прочего, эти макросы организуют доступ к компонентам карты сообщений базового класса посредством функций, которые анализируют карту сообщений производного класса. Это означает, что если класс является производным от CView, но не перехватывает сообщений, которые обычно перехватываются базовым классом, то сообщение будет перехвачено функцией класса CView, которая унаследована производным классом. Этот механизм наследования в карте сообщений работает параллельно с механизмом наследования C++, но независимо от него, и, таким образом, позволяет избежать многих сложностей, связанных с использованием виртуальных фуикций.

Подведем черту! Вы включаете в программу некоторый компонент карты сообщений и, когда такое сообщение возникает, функции, вызванные скрытым от вас циклом обработки сообщений, решают на основании этой таблицы, какой из объектов и какая из функций-членов этого объекта будет обрабатывать сообщеиие. Вот что в действительности происходит там, за кулисами.

Сообщения, которые перехватываются функциями MFC

Другое громадное преимущество MFC состоит в том, что в этой библиотеке уже имеются готовые классы, которые перехватывают и обрабатывают большинство распространенных сообщений, причем делают это безо всяких усилий со стороны разработчика программы. Например, вам не нужно заботиться об обработке таких сообщений, как вызов команды File⇨Save As. Классы MFC самостоятельно "отловят" это сообщение, выведут на экран диалоговое окно для ввода нового имени файла, обработают все манипуляции пользователя в этом окне, короче говоря, сделают для вас всю черновую работу и в конце вызовут разработанную уже вами функцию Serialize(), которая и запишет данные в файл. AppWizard, как правило, формирует пустую функцию Serialize(), в которую разработчик должен вставить необходимый текст. Подробнее о методике разработки функции Serialize() будет рассказано в главе 7. Таким образом, программисту необходимо вставлять в карту сообщений компоненты только для тех случаев, когда обработка некоторого сообщения в данном приложении отличается от общепринятой методики.

Как мастер ClassWizard помогает перехватывать сообщения

Читать карту сообщений в листинге программы совсем непросто, но зато ее очень просто формировать с помощью ClassWizard. В Visual C++ версии 6.0 существуют два способа включения компонента в карту сообщений — с помощью главного диалогового окна ClassWizard и с помощью одного из новых диалоговых окон, которые вставляют в программу обработчики сообщений или виртуальные функции. В этом разделе представлены подобного рода диалоговые окна для приложения ShowString, подробно рассматриваемого в главе 8.

Вкладки диалогового окна ClassWizard

Для того чтобы вывести на экран диалоговое окно мастера ClassWizard, нужно выбрать из меню View⇨ClassWizard или нажать <Ctrl+W>. ClassWizard имеет диалоговое окно с несколькими вкладками. На рис. 3.1 показана вкладка Message Maps (Карты сообщений). В верхней ее части имеется два раскрывающихся списка. В одном — Project — представлен проект, над которым вы в настоящее время работаете (в данном случае — ShowString), в другом — Class name — класс, карта сообщений которого редактируется (в данном случае — CShowStringApp). Информация о карте сообщений этого класса выведена в других полях вкладки.

Ниже этих однострочных полей расположена пара многострочных окон. В том, что слева, перечислены сам класс и все команды, которые может сформировать пользовательский интерфейс. Сами команды будут рассмотрены ниже в этой же главе, в разделе *Команды*. Когда в левом окне выделено имя класса, в правом перечислены все сообщения Windows, которые этот класс мог бы перехватывать. Кроме того, там же перечислены виртуальные функции, которые отвечают за обработку стандартных (наиболее распространенных) сообщений.

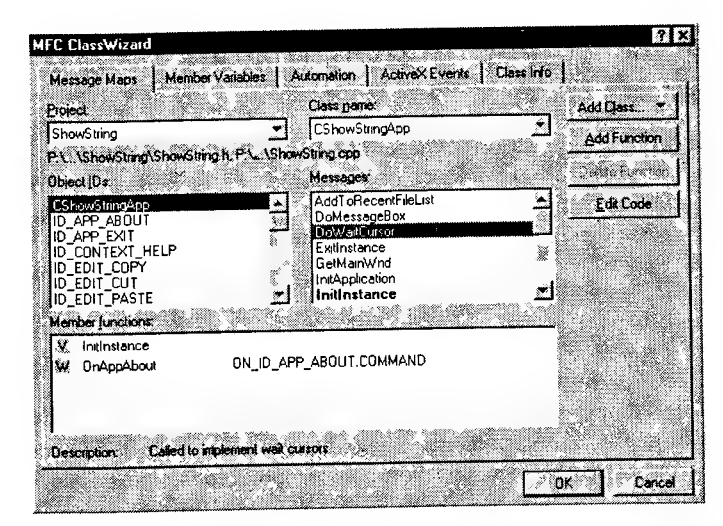


Рис. 3.1. ClassWizard *значительно упрощает программирование перехвата сообщений*

В правой верхней части окна находятся четыре кнопки, пользуясь которыми, можно включить новый класс в проект, новую функцию — в существующий класс, для того чтобы перехватывать выделенное в левом окне сообщение, удалить функцию, которая ответственна за обработку выделенного сообщения, или вывести на экран текст этой функции. Типовая методика следующая — нужно выбрать класс, сообщение и щелкнуть на кнопке Add Function с тем, чтобы добавить функцию, которая будет ответственна за обработку заданного сообщения. Ниже перечислены операции, которые будут выполнены после щелчка на Add Function.

- В конец файла текста программы включается заготовка ("скелет" — skeleton) функции.
- В файл текста программы, в ту его часть, где размещена карта сообщений, включается новый компонент карты.
- В файл заголовка также включается новый компонент карты сообщений.
- Обновляются списки сообщений и функций-членов в окнах вкладки.

После включения новой функции щелчок на Edit Code позволит наполнить созданную ClassWizard заготовку функции содержанием соответственно тому, как планируется обрабатывать данное сообщение. Того же результата можно достичь, сделав двойной щелчок на имени функции в списке Member Functions.

Этот список находится ниже окон списков Object IDs и Messages. В нем перечислены функции-члены текущего класса, которые связаны с определенными сообщениями. В данном случае имеются две такие функции.

- `InitInstance()`. Перегружает виртуальную функцию класса `CWinApp` — базового класса для `CShowStringApp`. Эта функция помечена символом `V` в списке, что означает — *виртуальная функция (Virtual function)*.
- `OnAppAbout()`. Перехватывает команду `ID_APP_ABOUT`; помечена символом `W` в списке, что означает — *сообщение окна (Window message)*.

Функция `InitInstance()` вызывается после запуска приложения. Вам нет необходимости углубляться в подробности ее работы — `ClassWizard` просто напоминает, что эта функция уже перегружена для данного приложения.

И наконец, ниже окна списка Member Functions выведено сообщение, напоминающее о назначении выделенного сообщения. В данном случае текст Called to implement wait cursors (Вызывается для установки курсора ожидания) — это описание виртуальной функции DoWaitCursor().

Добавление обработчиков сообщений Windows

В Visual C++ начиная с версии 5.0 предлагается новый способ перехвата сообщений. Вместо того чтобы вызвать ClassWizard и затем не забыть найти правильное имя класса в раскрывающемся списке, нужно просто щелкнуть *правой* кнопкой мыши на имени класса в окне ClassView и затем выбрать пункт Add Windows Message Handler (Добавление обработчиков сообщений Windows) контекстного меню. Диалоговое окно, которое в результате появится на экране, показано на рис. 3.2.

В этом диалоговом окне не показаны виртуальные функции, которые перечислены в главном окне ClassWizard. Глядя на это окно, легко сообразить, что данный класс перехватывает команду ID_APP_ABOUT, но не перехватывает обновление команды (command update). О командах и обновлениях команд будет более подробно рассказано далее в этой же главе. Для того чтобы добавить новую виртуальную функцию, нужно сделать двойной щелчок на имени класса в ClassView и выбрать пункт Add New Virtual Function (Добавление новой виртуальной функции) контекстного меню. Диалоговое окно, которое в результате появится на экране, показано на рис. 3.3.

На рис. 3.3 видно, что в классе CShowStringApp уже перегружена виртуальная функция InitInstance(), но существуют и другие виртуальные функции, которые можно перегрузить. Как и на вкладке, сообщение в самом низу окна напоминает вам о назначении каждой функции. Сам же текст сообщения — Called to implement wait cursors (Вызывается для установки курсора ожидания) — тот же, что и на вкладке (см. рис. 3.1).

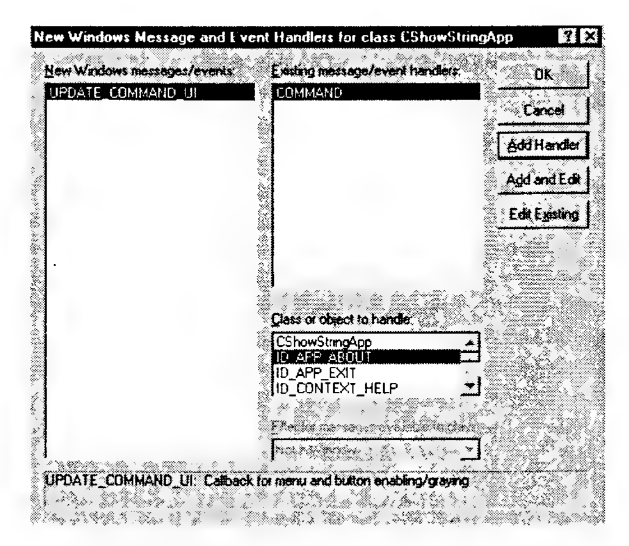


Рис. 3.2. Диалоговое окно New Windows Message and Event Handlers *— это новое средство для организации перехвата сообщений в проекте*

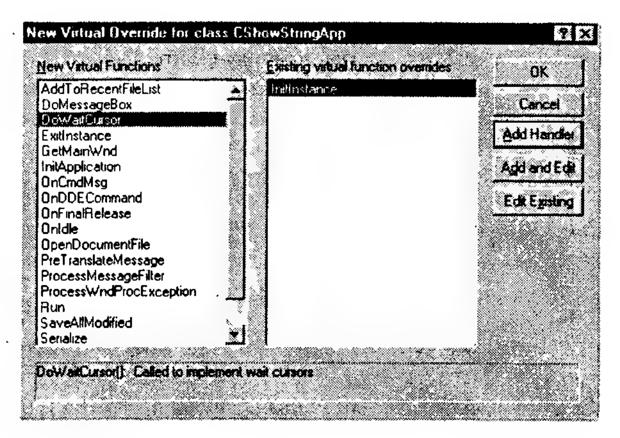


Рис. 3.3. Диалоговое окно New Virtual Override *упрощает реализацию виртуальных функций в проекте*

Какой класс должен перехватывать сообщение

Главный фокус в работе с картой сообщений и обработкой сообщений — решить, на какой класс возложить ответственность за перехват и обработку. Принять верное решение вы не сможете до тех пор, пока не будете четко представлять себе, для кого предназначены различные сообщения и команды типичного приложения. Обычно приходится выбирать что-нибудь из приведенного ниже списка.

- Активное представление (вид)
- Документ, представленный в нем
- Фрейм (рамка окна), который содержит активное представление
- Объект-приложение

Представления, документы и фреймы обсуждаются в следующей главе.

Список сообщений

Существует почти 900 различный сообщений, так что их исчерпывающий список просто не уместился бы в этой главе. Поскольку обычно для организации перехвата сообщений в приложении используется ClassWizard, представленный в нем список будет значительно короче (ведь в него отбираются только те сообщения, которые подходят для выбранного класса). Отнюдь не каждое окно может получить то или иное сообщение. Например, только класс, являющийся наследником CListBox, может получить сообщение типа LB_SETSEL, которое заставляет элемент управления типа список передвинуть подсветку на некоторый элемент списка. Префикс в имени сообщения указывает тип окна, для которого предназначено сообщение или которое его породило. Эти префиксы перечислены в табл. 3.1.

Таблица 3.1. Префиксы сообщений и типы окон

Префикс	Тип окна
ABM, ABN	Панель приложения (Appbar)
ACM, ACN	Элемент управления типа анимация (Animation control)
BM, BN	Кнопка (button)
CB, CBN	Поле со списком (combo box)
CDM, CDN	Стандартное диалоговое окно (common dialog box)
CPL	Приложение Control Panel (Панель управления)
DBT	Любое приложение (сообщение о переназначении устройства)
DL	Окно списка перетаскивания (drag list box)
DM	Диалоговое окно
EM, EN	Текстовое поле (edit box)
FM, FMEVENT	Менеджер файлов (File Manager)
HDM, HDN	Элемент управления типа заголовок (header control)
HKM	Элемент управления типа функциональная клавиша (HotKey control)
IMC, IMN	Окно типа IME
LB, LBN	Элемент управления типа список (list box)
NM	Любое родительское окно (сообщение с кодом извещения)
PBM	Элемент управления типа линейный индикатор (progress bar)
PBT	Любое приложение, оповещение о состоянии батарейного питания (battery power broadcast)
PSM, PSN	Вкладка свойств (property sheet)
SB	Строка состояния (status bar)
SBM	Полоса прокрутки (scroll bar)
STM, STN	Элемент управления типа статическая надпись (static control)
TB, TBN	Панель инструментов
TBM	Элемент управления типа линейный регулятор (track bar)
TCM, TCN	Элемент управления типа вкладка (tab control)
TTM, TTN	Контекстное окно указателя (ToolTip)
TVM, TVN	Просмотровое окно дерева (tree view)
UDM	Элемент управления типа инкрементный регулятор (UpDown control)
WM	Окно как таковое (generic window)

Какая разница, скажем, между BM- и BN-сообщением? BM-сообщение — это сообщение, *направленное объекту-кнопке*, например "Действуй так, как будто на тебе щелкнули". А BN-сообщение — это сообщение с кодом извещения, поступающего *от объекта-кнопки* окну, в котором эта кнопка находится и которое является "владельцем" кнопки. Это сообщение может, например, гласить: "Ой, на мне щелкнули!". То же самое справедливо для всех других модификаций сообщений, префиксы которых завершаются литерой M или N в приведенной выше таблице.

Иногда префикс не заканчивается буквой M. Например, CB — это префикс для сообщений от объекта — поля со списком, в то время как CBN является префиксом сообщения с кодом извещения, которое поле со списком передает окну-владельцу. Например, CBN_SELCHANGE — это сообщение от поля со списком, извещающее "родителя" о том, что пользователь выбрал другой элемент списка.

Команды

Что такое команда? Это сообщение специального типа, которое формируется в тех случаях, когда пользователь выбирает пункт меню, щелкает на кнопке или каким-либо другим способом дает системе понять, что ему что-то от нее нужно. В прежних версиях Windows и выбор из меню, и щелчок на кнопке формировали сообщение WM_COMMAND. Наступили новые времена, и теперь только выбор из меню порождает сообщение WM_COMMAND, а щелчок на кнопке или выбор в списке порождает сообщение WM_NOTIFY с кодом извещения от элемента управления. Команды и коды извещений проскакивают через операционную систему точно так же, как и любые другие сообщения, но только до тех пор, пока не попадут в функцию OnWndMsg(). Здесь заканчиваются владения Windows, и в дело вступают средства маршрутизации команд, имеющиеся в MFC.

Все сообщения команд содержат в качестве первого параметра идентификатор ресурса — выбранный пункт меню или кнопку, на которой щелкнули. Этот идентификатор ресурса присваивается в соответствии со стандартом на форматы подобного рода идентификаторов, например для пункта Save As меню File идентификатор будет иметь вид ID_FILE_SAVE.

Маршрутизация команд — это механизм, используемый функцией OnWndMsg() для передачи команд и кодов извещения объектам, которые не могут получать *сообщения*. Получать *сообщения* могут только объекты классов-наследников CWnd, а все объекты классов, порожденных от CCmdTarget, включая CWnd и CDocument, могут получать *команды* и *извещения*. Это означает, что класс, который наследует CDocument, может иметь карту сообщений, причем в ней не должно быть ни одного компонента, соответствующего сообщению, а только компоненты для команд и извещений. Тем не менее этот фрагмент программы по-прежнему называется *картой сообщений*.

Каким же образом все-таки команды и извещения передаются классу? Посредством механизма маршрутизации команд. (Это довольно занудная процедура, так что, если вас не интересуют подробности ее реализации, можете со спокойной совестью перейти к следующему абзацу.) Функция OnWndMsg() вызывает CWnd::OnCommand() или CWnd::OnNotify(). Функция OnCommand() исследует всю подноготную команды (например, не нужно ли будет заблокировать этот пункт меню после того, как пользователь его выбрал, но перед тем, как соответствующий фрагмент программы будет выполнен) и затем вызывает OnCmdMsg(). Функция OnNotify() анализирует другие условия и затем также вызывает OnCmdMsg(). Функция OnCmdMsg() является виртуальной, а это означает, что различные классы, для которых предназначены различные команды, имеют разные реализации этой функции. Фрейм пересылает команду представлению или документу, который он (фрейм) содержит.

Вот так выполняется процесс, начинающийся как сообщение и заканчивающийся функцией-членом объекта, который не является окном, и, таким образом, не может в действительности перехватывать сообщения.

Зачем вам все это знать? Даже если вы не знаете, как происходит маршрутизация, вам придется позаботиться о правильном выборе классов, которые будут обрабатывать все события, которые могут произойти в разрабатываемом приложении. Если пользователь изменяет размеры окна, посылается сообщение WM_SIZE, и вам, возможно, понадобится изменить масштаб изображения или выполнить еще что-нибудь с представлением в приложении. Если пользователь выбирает некоторый пункт меню, формируется команда, а это означает, что класс документа должен что-то сделать в ответ на нее. Примеры реализации на практике этого подхода рассматриваются в следующей главе.

Обновление команд

Наша краткая экскурсия по подземельям MFC, в которых мы попробовали познакомить вас с тем, как связываются действия пользователя с текстом программы, завершается. Остался последний зал, в котором можно будет узнать, как программа выполняет блокировку определенных пунктов меню или кнопок в соответствии с контекстом задачи. Этот процесс назван *обновлением команд* (command updating).

Представьте себе на минуточку, что вы разрабатываете приложение и у вас возникла идея блокировать некоторые команды меню, чтобы показать, что они в данный момент недоступны. Реализовать эту прекрасную идею можно двумя способами.

Один состоит в том, чтобы организовать огромную таблицу, элементами которой будут все имеющиеся в приложении пункты меню, каждому из которых сопоставлен флаг. Состояние флага — TRUE или FALSE — указывает, доступен ли этот пункт меню. Как только возникает необходимость вывести меню на экран, можно быстренько просмотреть таблицу и все сразу станет ясно. При любой операции, которая может повлечь за собой изменения в статусе какого-либо пункта меню, таблица обновляется. Все это в совокупности называется *подходом непрерывного обновления* (continuous-updating approach).

Другой подход состоит в том, чтобы, не имея такой таблицы, перед каждым выводом меню на экран анализировать все условия, которые влияют на возможную блокировку. Он называется *подходом обновления по требованию* (update-on-demand approach). Именно такой подход и реализован в Windows. До широкого внедрения идей объектно-ориентированного программирования в разработку Windows-приложений такой анализ выполнялся следующим образом: система посылала сообщение WM_INITMENUPOPUP, которое означало: "Я готовлюсь вывести на экран меню", огромный оператор switch в функции WinProc()перехватывал это сообщение и быстренько разрешал или блокировал определенные пункты меню. Эта методика абсолютно противоречит главным идеям объектно-ориентированного программирования, которые требуют, чтобы разные части информации хранились в разных объектах и не морочили голову остальной программе — были от нее скрыты.

Когда наступает время выводить на экран меню, конкретные объекты "знают", нужно ли блокировать связанный с ними пункт меню. Например, объект класса документа знает, был ли он модифицирован после последнего сохранения, и решает, стоит ли блокировать пункт Save меню File. Объект класса представления знает, есть ли выделенный фрагмент текста, и может решить, как поступить с пунктами Cut и Copy меню Edit. Все это означает, что комплексная задача блокировки пунктов меню в соответствии с контекстом приложения распределяется между различными объектами приложения, а не возлагается на главную вызывающую подпрограмму WndProc().

Подход, реализованный в MFC, состоит в том, чтобы использовать небольшой объект класса CCmdUI (класс интерфейса с командами пользователя — command user interface) и предоставить ему возможность перехватывать любые сообщения CN_UPDATE_COMMAND_UI. Организовать такой перехват можно, добавив (или предоставив возможность ClassWizard добавить) макрос ON_UPDATE_COMMAND_UI в карту сообщений. За ширмой макроса происходит следующее: операционная система по-прежнему посылает сообщение WM_INITMENUPOPUP, а затем в дело вступают базовые классы MFC, такие как CFrameWnd. Они формируют объект класса CCmdUI, устанавливают значения его членов-переменных соответственно первому пункту меню и вызывают один из методов этого класса — DoUpdate(). Затем DoUpdate() посылает сообщение CN_UPDATE_COMMAND_UI, направляя его самому себе, так как обработчиком сообщений указан объект CCmdUI. Затем тот же самый объект CCmdUI перенастраивается соответственно второму пункту меню и т.д., пока все меню не будет подготовлено к выводу. Объект класса

CCmdUI также используется для блокировки или разблокировки командных кнопок и других элементов управления по той же технологии.

Класс CCmdUI имеет следующие функции-члены.

- Enable(). Принимает аргумент TRUE или FALSE. Блокирует элемент интерфейса пользователя, если передан аргумент FALSE, в противном случае делает элемент доступным.
- SetCheck(). Включает или выключает элемент управления.
- SetRadio(). Включает или выключает элемент управления как принадлежащий к группе зависимых переключателей, из которых только один может быть включен.
- SetText(). Устанавливает текст надписи пункта меню или кнопки, если элемент управления — кнопка.
- DoUpdate(). Формирует сообщение.

Как правило, выбор, какая из перечисленных функций-членов нужна вам для определенных целей, очевиден. Ниже приведена упрощенная версия карты сообщений объекта класса CWhoisView, производного от CFormView, который выводит информацию на экран для пользователя. Соответствующая экранная форма имеет несколько текстовых полей, и пользователь может вставлять текст в одно из них. Карта сообщений содержит компонент перехвата обновления для команды ID_EDIT_PASTE, что в тексте программы выглядит следующим образом:

```
BEGIN_MESSAGE_MAP(CWhoisView, CFormView)
 ...
 ON_UPDATE_COMMAND_ID(ID_EDIT_PASTE, OnUpdateEditPaste)
 ...
END_MESSAGE_MAP()
```

Функция OnUpdateEditPaste(), которая перехватывает обновление, выглядит следующим образом:

```
void CWhoisView::OnUpdateEditPaste(CCmdUI* pCmdUI)
{
  pCmdUI->Enable(::IsClipboardFormatAvailable(CF_TEXT));
}
```

Здесь вызывается функция API ::IsClipboardFormatAvailable(), которая проверяет, есть ли текст в системном буфере Clipboard. Дело в том, что другое приложение может загрузить из Clipboard изображение или другую нетекстовую информацию, но данное приложение такими возможностями не располагает. Поэтому, если в Clipboard содержится не текст, текстовое поле в экранной форме блокируется. Большинство функций обновления команд выглядит аналогично — они вызывают Enable()с аргументом, который формируется в результате вызова некоторой функции, возвращающей TRUE или FALSE, или — другой вариант — аргумент является просто логическим выражением. Обработчики обновления команд должны работать очень быстро, поскольку с момента, когда пользователь вызвал на экран меню, щелкнув в нужном месте окна, нужно успеть пропустить пять-десять циклов обработки и только после этого обновленное меню будет выведено на экран.

Как ClassWizard помогает перехватывать команды и их обновления

В диалоговом окне ClassWizard, представленном на рис. 3.1, в списке Object IDs выделено имя класса. Ниже имеются идентификаторы всех ресурсов (меню, панелей инструментов, элементов управления и т.д.), которые могут формировать команду или сообщение при условии, что объект данного класса присутствует на экране. Если вы выделите один из них, то список связанных с ним сообщений Messages станет значительно короче, как это видно на рис. 3.4.

С каждым идентификатором ресурса связаны только две строки в списке сообщений — COMMAND и UPDATE_COMMAND_UI. Выбор первой позволяет добавить в класс функцию, которая будет обрабатывать либо ту команду меню, которую выбрал пользователь, либо щелчок на кнопке, т.е. в конечном счете обрабатывать команду. Выбор второй позволяет добавить в класс функцию, которая задает состояние пункта меню, кнопки или любого другого элемента управления перед тем, как операционная система соберется вывести его на экран, т.е. выполнить обновление команды. (Строка COMMAND выведена на рис. 3.4 полужирным шрифтом, поскольку в классе CShowStringApp перехват этой команды уже запрограммирован.)

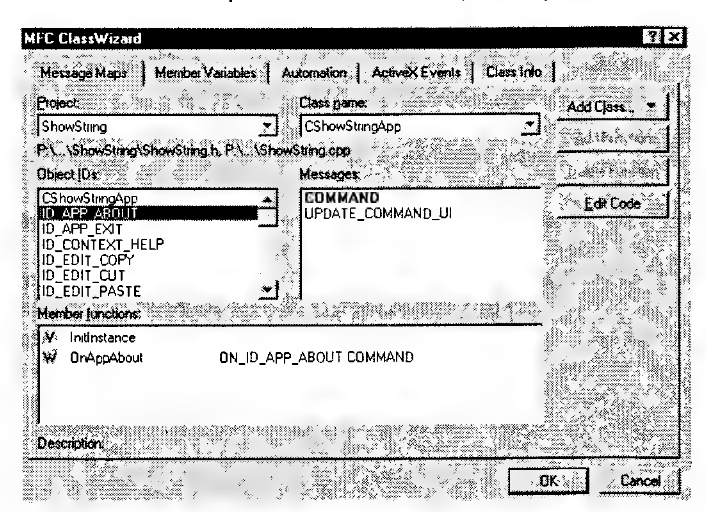


Рис. 3.4. ClassWizard позволяет программировать перехват или обновление команд

Если вы считаете нужным ввести новую функцию для перехвата команды или обновления, щелкните на кнопке Add Function. Это включит в процесс разработки еще один этап — ClassWizard предоставит вам возможность изменять имя функции, которое он сформировал по стандартной схеме (рис. 3.5). Эта возможность оставлена для самых привередливых, поскольку стандартная схема формирования имени, как правило, не вызывает возражений даже у опытных программистов, давно имеющих дело с MFC. Имя функции обработки команды начинается с On. Оставшаяся часть формируется следующим образом — удаляется ID и символы подчеркивания из идентификатора ресурса, а каждое слово пишется строчными литерами, кроме первого символа. Имена обработчиков обновления команд начинаются с OnUpdate, а далее используется то же преобразование идентификатора ресурса. Например, функция, которая перехватывает команду от ID_APP_EXIT, будет называться OnAppExit, а функция, которая обновляет ID_APP_EXIT, будет называться OnUpdateAppExit.

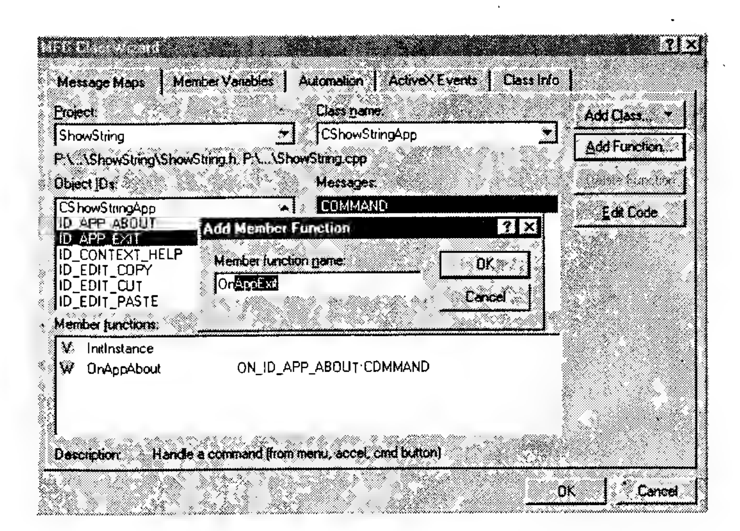


Рис. 3.5. ClassWizard позволяет изменить сформированное им имя функции обработки команды или обновление команды, что, однако, делать не рекомендуется

Далеко не каждая команда нуждается в обработчике обновления. Объект класса фрейма окна выполняет некоторую работу в части блокировки элементов управления самостоятельно, безо всяких указаний на то со стороны разработчика. Пусть, скажем, у вас есть меню Network (Сеть), а в нем — пункт Sent (Послать). Команда этого пункта меню перехватывается объектом класса документа. Если же в приложении не открыт ни один документ, этот пункт меню будет заблокирован главным окном приложения, причем блокировка организуется безо всяких усилий с вашей стороны. Для многих команд этого вполне достаточно, т.е. команда блокируется, если объект, который должен ее обрабатывать, не существует. Для других же, которые могут иметь смысл только в случае, если что-то выбрано или выделено, схема блокировки значительно сложнее. Вот здесь и нужно участие разработчика в программировании процесса обновления команды.

Пустая страница

ЧАСТЬ II

Программирование вывода информации в приложении

В этой части...

ГЛАВА 4

Документы и представления

В этой главе...

Что такое класс документа

Что такое класс представления

Создание приложения Rectangles

Другие классы представления

Шаблоны документов, окна представления и окна

Что такое класс документа

Формируя текст программы приложения с помощью AppWizard, вы имеете возможность оснастить свое детище всеми модными аксессуарами коммерческого продукта для Windows 95 — панелью инструментов, строкой состояния, контекстным окном указателя, разнообразными меню и даже окном сообщения об авторских правах. Однако несмотря на все эти "примочки" полезность такого приложения равна нулю. Для того чтобы создать приложение, которое не только хорошо выглядит на экране, но и делает что-то полезное, вам волей-неволей придется вмешаться в текст программы, которую подготовил AppWizard. Это может быть просто или не очень просто, или даже очень сложно — все зависит от того, что же в действительности должно делать приложение и как оно должно при этом выглядеть.

Возможно, наиболее существенным изменениям подвергнется часть подготовленной AppWizard программы, связанная с документом — информацией, которую пользователь может сохранять в процессе работы с приложением и затем считывать, — и с представлением — средствами представления этой информации пользователю в процессе выполнения приложения. Положенная в основу MFC концепция *документ/представление* позволяет отделить данные от средств, с помощью которых пользователь имеет возможность просмотреть эти данные и манипулировать ими. Короче говоря, объекты-документы ответственны за хранение, загрузку и выгрузку данных, а объекты-представления, которые подчас представляют собой те же окна, позволяют пользователю просматривать данные на экране и редактировать их соответственно логике работы приложения. В этой главе вы познакомитесь с основами работы MFC в части реализации концепции *документ/представление*.

Создавая SDI- и MDI-приложения, AppWizard изначально закладывает в них средства, ориентированные на реализацию концепции *документ/представление*. Это означает, что AppWizard формирует класс, производный от CDocument, и передает ему определенные задачи. Он также формирует класс представления, производный от CView, которому передает другие задачи. Давайте закажем AppWizard простейшее приложение и посмотрим, что он нам выдаст.

Выберите File⇨New, затем вкладку Projects. Установите имя проекта App1 и соответствующие каталоги для файлов проекта. Проверьте, чтобы был выбран вариант MFC AppWizard (exe) в левом окне. Щелкните на OK.

Пройдитесь по всем диалоговым окнам AppWizard, заказывая параметры в соответствии с приведенным ниже списком и каждый раз щелкая на Next.

- **Этап 1**. Выберите Multiple documents.
- **Этап 2**. Не меняйте настроек, предлагаемых AppWizard по умолчанию.
- **Этап 3**. Не меняйте настроек, предлагаемых AppWizard по умолчанию.
- **Этап 4**. Сбросьте все флажки, кроме Printing and print preview (Печать и предварительный просмотр распечатки).
- **Этап 5**. Не меняйте настроек, предлагаемых AppWizard по умолчанию.
- **Этап 6**. Не меняйте настроек, предлагаемых AppWizard по умолчанию.

На последнем этапе щелкните на кнопке Finish, и параметры выполненной настройки будут выведены в окне New Project Information (Информация о новом проекте). После щелчка на кнопке OK ClassWizard сформирует проект. В окне ClassView просмотрите список классов приложения. Создано шесть классов: CAboutDlg, CApp1App, CApp1Doc, CApp1View, CChildFrame и CMainFrame.

Класс CApp1Doc представляет документ и содержит структуру данных, которыми может оперировать приложение. Организовать хранение данных в классе можно включением в него

соответствующих членов-переменных. Текст файла заголовка, который AppWizard сформировал для класса CApp1Doc, представлен в листинге 4.1.

Листинг 4.1. APP1DOC.h **— главный файл заголовка для класса** CApp1Doc

```
// App1Doc.h : интерфейс класса CApp1Doc.
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_APP1DOC_H__43BB481D_64AE_11D0_9AF3_0080C81A397C__INCLUDED_)
#define AFX_APP1DOC_H__43BB481D_64AE_11D0_9AF3_0080C81A397C__INCLUDED_


class CApp1Doc : public CDocument
{
protected: // Создаются только в случае сохранения-восстановления.
   CApp1Doc();
   DECLARE_DYNCREATE(CApp1Doc)

// Атрибуты.
public:

// Операции.
public:

// Перегрузка.
   // Перегрузка виртуальных функций, сформированная ClassWizard.
   //{{AFX_VIRTUAL(CApp1Doc)
   public:
   virtual BOOL OnNewDocument();
   virtual void Serialize(CArchive& ar);
   //}}AFX_VIRTUAL

// Реализация.
public:
   virtual ~CApp1Doc();
#ifdef _DEBUG
   virtual void AssertValid() const;
   virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Карта сообщений.
protected:
   //{{AFX_MSG(CApp1Doc)
      // ВНИМАНИЕ!!  Здесь ClassWizard будет добавлять и
      //    удалять функции-члены.
      //    НЕ РЕДАКТИРУЙТЕ текст в этих блоках!
   //}}AFX_MSG
   DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ будет вставлять дополнительные
//     объявления непосредственно
//     перед предыдущей строкой.

#endif // !defined(AFX_APP1DOC_H__43BB481D_64AE_11D0_9AF3_0080C81A397C__INCLUDED_)
```

Почти в самом начале листинга есть секция Атрибуты, за которой следует ключевое слово public. Именно здесь вам и нужно будет вставить объявления членов-переменных, в которых планируется хранить данные приложения. В приложении, которое будет рассмотрено дальше в этой главе, нужно будет сохранять массив объектов класса CPoint. Этот массив объявляется как член класса документа:

```
// Атрибуты.

public:

 CPoint m_points[100];
```

CPoint — это класс MFC, который инкапсулирует информацию, имеющую отношение к точке на экране, в частности — координаты этой точки x и y.

В тексте файла заголовка обратите внимание также на то, что класс CApp1Doc имеет две виртуальные функции-члены — OnNewDocument() и Serialize(). MFC вызывает функцию OnNewDocument(), как только пользователь выберет команду File⇨New (или соответствующую пиктограмму на панели инструментов, если таковая присутствует в приложении). Эту функцию можно использовать для выполнения всех инициализаций, необходимых новой порции данных. В SDI-приложении в таком случае закрывается открытый документ и новый пустой документ загружается в тот же самый объект класса. В MDI-приложении открывается новый пустой документ (создается новый экземпляр класса документа) в дополнение к уже существующему. Функция Serialize() используется для загрузки в файл и выгрузки из него данных, хранящихся в членах-переменных объекта класса документа. Об этом более подробно будет рассказано в главе 7.

Что такое класс представления

Как уже упоминалось, класс представления отвечает за вывод на экран данных, хранящихся в объекте класса документа, и позволяет пользователю модифицировать эти данные. Объект класса представления содержит указатель на объект класса документа, который используется для доступа к членам-переменным этого класса, где собственно и хранятся данные. Листинг 4.2 содержит текст файла заголовка, который AppWizard сформировал для класса CApp1View.



Большинство программистов, имеющих дело с MFC, включают в класс документа открытые (public) члены с тем, чтобы не затруднять доступ к ним из объекта класса представления. Классический объектно-ориентированный подход, однако, требует включать в класс закрытые (private) или защищенные (protected) члены-переменные и открытые члены-функции считывания и модификации этих переменных.

Листинг 4.2. APP1VIEW.H — главный файл заголовка для класса CApp1View

```
// App1View.h : интерфейс класса CApp1View.
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_APP1VIEW_H__43BB481F_64AE_11D0_9AF3_0080C81A397C__INCLUDED_)
#define AFX_APP1VIEW_H__43BB481F_64AE_11D0_9AF3_0080C81A397C__INCLUDED_

class CApp1View : public CView
{
protected: // Создавать только в случае сохранения-ввода.
   CApp1View();
```

```
  DECLARE_DYNCREATE(CApp1View)

// Атрибуты.
public:
  CApp1Doc* GetDocument();

// Операции.
public:

// Перегрузка.
  // Виртуальная функция, созданная ClassWizard, перегружает
  //{{AFX_VIRTUAL(CApp1View).
  public:
  virtual void OnDraw(CDC* pDC);  // Перегружена для прорисовки
      // в этом представлении.
  virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
  protected:
  virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
  virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
  virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
  //}}AFX_VIRTUAL

// Реализация.
public:
  virtual ~CApp1View();
#ifdef _DEBUG
  virtual void AssertValid() const;
  virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Сформированные функции карты сообщений.
protected:
  //{{AFX_MSG(CApp1View)
    // ВНИМАНИЕ!! Здесь ClassWizard будет добавлять и
    //   удалять функции-члены.
    //   НЕ РЕДАКТИРУЙТЕ текст в этих блоках!
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()
};

#ifndef _DEBUG
inline CApp1Doc* CApp1View::GetDocument()
  { return (CApp1Doc*)m_pDocument; }
#endif

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ будет вставлять дополнительные
//    объявления непосредственно
//    перед предыдущей строкой.

#endif // !defined(AFX_APP1VIEW_H__43BB481F_64AE_11D0_9AF3_0080C81A397C__INCLUDED_)
```

Почти в самом начале текста имеется секция открытых атрибутов, в которой объявлена функция GetDocument(), возвращающая указатель на объект класса CApp1Doc. Если, работая с классом представления, вы пожелаете получить указатель на объект класса документа, нужно будет вызвать эту функцию и она вернет вам требуемый указатель. Например, для того чтобы добавить объект класса CPoint в массив таких объектов, который является членом класса документа, можно использовать следующий оператор:

```
GetDocument()->m_points[x] = point;
```

Можно сделать то же самое, запомнив указатель, возвращаемый GetDocument(), в локальной переменной, а затем уже использовать ее для доступа к данным документа:

```
pDoc = GetDocument();
pDoc->m_points[x] = point;
```

Второй вариант имеет смысл, если вы будете неоднократно использовать сохраненный указатель в функции или если выражение в форме GetDocument()->*переменная* кажется вам сложным для восприятия в тексте программы.

На заметку В распространяемой (release) версии приложения функция GetDocument() объявлена как *встроенная* (inline). В смысле производительности программы нет никакой разницы в обеих формах обращения к членам объекта документа. Но в смысле читабельности текста программы второй вариант, несомненно, имеет преимущество. Встроенные функции расширяются в скомпилированной программе так же, как и макросы, но, в отличие от макросов, компилятор при работе со встроенными функциями выполняет проверку типов аргументов. Существуют и другие преимущества замены макросов, где это возможно, встроенными функциями. Это более подробно обсуждается в приложении А.

Обратите внимание на то, что как класс представления, так и класс документа перегружают часть функций-членов своих базовых классов. Как вы вскоре убедитесь, функция OnDraw(), которая является одной из важнейших среди всех виртуальных функций, является именно тем инструментом, с помощью которого производится рисование в окне приложения. Что касается других функций, то MFC вызывает PreCreateWindow() перед тем, как создается и присоединяется к объекту класса окон MFC окно Windows. Эта функция дает возможность модифицировать такие атрибуты окна, как положение и размер. Эти две функции обсуждаются более детально в главе 5. Функция OnPreparePrinting() используется для модификации диалогового окна Print перед его выводом на экран. Функция OnBeginPrinting() дает шанс создать GDI-объект, такой как кисть или перо, который необходим для выполнения некоторых задач в процессе печати. И наконец, в функции OnEndPrinting() можно уничтожить любой объект, созданный функцией OnBeginPrinting(). Все три упомянутые функции детально рассматриваются в главе 6.

На заметку На первых порах, когда вы только начинаете работать с MFC, многим кажется очень непривычной разница между экземпляром класса MFC и элементом Windows, который он представляет. Например, когда вы создаете объект класса фрейма MFC, в действительности создаются два объекта — MFC-объект, который имеет члены-функции и переменные, и окно Windows, которым можно манипулировать, используя функции MFC-объекта. Элемент Windows ассоциируется с соответствующим классом, но сам по себе также представляет некую сущность.

Создание приложения Rectangles

Теперь, познакомившись с классами представления и документа, желательно увидеть, как все это выглядит на практике. В этом разделе мы будем создавать приложение Rectangles, которое продемонстрирует манипуляции документами и представлениями. Когда вы в первый раз запустите это приложение, оно выведет на экран пустое окно. После любого щелчка на поле окна в этом месте будет нарисован маленький квадратик. Окно можно растягивать или сжимать, сворачивать в пиктограмму или распахивать на весь экран, но в любом случае квадратики будут перерисовываться в том же месте (относительно координат окна), где вы щелкнули. Это обеспечивается сохранением массива координат точек в документе и использованием этого массива в представлении.

Первая задача — сформировать "заготовки" файлов программы с помощью AppWizard. Для этого в процессе настройки AppWizard используйте перечисленные ниже параметры.

(Использование AppWizard обсуждалось в главе 1.) Когда закончите, появится окно New Project Information, в котором суммированы выполненные настройки. Щелкните в нем на кнопке OK и тем самым запустите процесс создания файлов проекта.

Имя диалогового окна	Выбираемая опция
New Project	Имя проекта recs. Установите также пути к каталогам, в которых собираетесь сохранять файлы проекта. Остальные опции оставьте в том состоянии, которое AppWizard предлагает по умолчанию
Step 1 of 6	Выберите Single document
Step 2 of 6	Не меняйте настроек, предлагаемых AppWizard по умолчанию
Step 3 of 6	Не меняйте настроек, предлагаемых AppWizard по умолчанию
Step 4 of 6	Сбросьте все флажки, кроме Printing and print preview
Step 5 of 6	Не меняйте настроек, предлагаемых AppWizard по умолчанию
Step 6 of 6	Не меняйте настроек, предлагаемых AppWizard по умолчанию

Теперь, когда у вас есть заготовка файлов проекта, наступило время включить в тексты программ все необходимое для классов документов и представлений с тем, чтобы создать нечто действительно работающее. Это приложение будет рисовать на экране квадратики в представлении и сохранять координаты их базовых точек в документе (не слишком полезное действо, но уже что-то большее, чем просто переключение между окнами).

Следуйте приведенным ниже инструкциям и добавьте в файлы реализации класса документа операторы, необходимые для обработки данных приложения. Данные, напомним еще раз, — это массив объектов класса CPoint, которые определяют координаты базовых точек квадратиков на поле окна представления.

1. Щелкните на корешке вкладки ClassView (Просмотр классов) с тем, чтобы в левой части экрана открылось окно ClassView.
2. Разверните список классов приложения Recs. Для этого нужно щелкнуть на значке + перед ним.
3. Щелкните правой кнопкой мыши на классе CRecsDoc. Появится контекстное меню, из которого нужно выбрать пункт Add Member Variable (Добавить члены-переменные).
4. Заполните поля диалогового окна Add Member Variable. В поле Variable Type (Тип переменной) введите CPoint. В поле Variable Declaration (Объявление переменной) введите m_points[100]. Проверьте, чтобы был выбран переключатель Public (Открытые). Щелкните на OK.
5. Опять щелкните правой кнопкой мыши на классе CRecsDoc и снова выберите пункт Add Member Variable.
6. В поле Variable Type введите UINT. В поле Variable Declaration введите m_pointIndex. Проверьте, чтобы был выбран переключатель Public. Щелкните на OK.
7. Разверните список членов класса CRecsDoc, щелкнув на значке + перед ним. Вы увидите, что два новых члена, только что добавленных в класс, перечислены в списке.

Массив m_points[] будет хранить координаты опорных точек квадратиков, а m_pointIndex будет хранить индекс очередного свободного элемента этого массива (и соответственно — количество занесенных в него пар координат).



Если у вас есть опыт программирования на C++ и вы предпочитаете не обращаться к ClassView, поступите следующим образом. Откройте файл RecsDoc.h с помощью File⇨View и вставьте (после строки public:) две строки объявления этих переменных:

```
UINT m_pointIndex;
CPoint m_points[100];
```

Теперь нужно позаботиться об инициализации этих переменных и затем спокойно использовать их в процессе вывода на экран представления документа. Приложение, которое использует MFC, применяется для этих целей функцией DnNewDocument(). Она вызывается автоматически после запуска приложения и при выполнении пользователем команды File⇨New.

Список членов класса CRecsDoc должен по-прежнему находиться в окне ClassView. Дважды щелкните на элементе DnNewDocument() в списке, и можно будет приступить к редактированию текста этой функции. Образцом вам послужит листинг 4.3. Удалите комментарии, оставленные AppWizard, и инициализируйте m_pointIndex, присвоив ему значение 0.

Листинг 4.3. Файл RecsDoc.cpp — CRecsDoc::OnNewDocument()

```
BDDL CRecsDoc::DnNewDocument()
{
  if (!CDocument::DnNewDocument())
    return FALSE;

   m_pointIndex = 0;

  return TRUE;
}
```

Нет никакой необходимости инициализировать массив, поскольку для записи в него будет использован уже инициализированный индекс. Так что на этом модификацию класса документа можно считать успешно завершившейся. Как вы увидите позже, в главе 7, понадобится сделать еще несколько незначительных исправлений, если вы хотите, чтобы данные нормально сохранялись в файле документа. Поскольку сейчас наша цель — сконцентрироваться на совместной работе документа и его представления, в приложении Recs эти детали мы вносить не будем.

Теперь перейдем к классу представления. Он будет пользоваться данными из документа для прорисовки квадратиков на экране. Детально рассматривать процесс рисования мы будем в главе 5. Здесь же вполне достаточно знать, что функция DnDraw() созданного класса представления берет на себя все заботы по прорисовке. Разверните класс CRecsView в окне ClassView и сделайте двойной щелчок на элементе OnDraw(). Используя листинг 4.4 как образец, удалите комментарии, оставленные AppWizard, и добавьте операторы, которые прорисовывают квадратик, используя в качестве опорной точку с координатами, заданными в массиве.

Листинг 4.4. Файл RecsView.cpp — CRecsView::OnDraw()

```
void CRecsView::DnDraw(CDC* pDC)
{
  CRecsDoc* pDoc = GetDocument();
  ASSERT_VALID(pDoc);

   UINT pointIndex = pDoc->m_pointIndex;

   for (UINT i=D; i<pointIndex; ++i)
   {
      UINT x = pDoc->m_points[i].x;
      UINT y = pDoc->m_points[i].y;
      pDC->Rectangle(x, y, x+20, y+20);
   }
}
```

Теперь можно считать, что модификация заготовок файлов, которые создал AppWizard, практически завершена. Вы добавили члены-переменные в класс документа, инициализировали их в функции-члене этого класса DnNewDocument() и использовали эти переменные в функции-члене класса представления OnDraw(). Осталась ерунда — позволить пользователю пополнять массив новыми координатами. Для этого нужно с помощью ClassWizard организовать перехват сообщения, формируемого после щелчка кнопкой мыши (от этом шла речь в

главе 3), и затем в функцию обработки этого щелчка добавить необходимые операторы. Следуйте за нами.

1. Выберите View⇨ClassWizard. Появится диалоговое окно ClassWizard.
2. Проверьте, чтобы в окнах Class Name и Object IDs были выбраны CRecsView. Теперь сделайте двойной щелчок на WM_LBUTTONDOWN в окне Message и включите в класс функцию-член обработки щелчка. Теперь, как только приложение получит сообщение WM_LBUTTONDOWN, оно будет вызывать функцию OnLButtonDown().
3. Щелкните на кнопке Edit Code, и, таким образом, вы сможете приступить к редактированию текста функции OnLButtonDown() в окне редактора кода. После редактирования она должна выглядеть так, как показано в листинге 4.5.

Листинг 4.5. Файл RecsView.cpp — CRecsView::OnLButtonDown()

```
void CRecsView::OnLButtonDown(UINT nFlags, CPoint point)
{
    CRecsDoc *pDoc = GetDocument();

   // Предупреждает выход за пределы объявленной длины
   // массива - 100 элементов.
    if (pDoc->m_pointIndex == 100)
       return;

    //Запоминается точка, в которой произведен щелчок.
    pDoc->m_points[pDoc->m_pointIndex] = point;
    pDoc->m_pointIndex++;

    pDoc->SetModifiedFlag();
    Invalidate();

   CView::OnLButtonDown(nFlags, point);
}
```

Функция OnLButtonDown() в таком варианте вписывает в массив точек объекта класса документа пару координат указателя мыши после щелчка левой кнопкой мыши на поле окна представления. После этого m_pointIndex увеличивается на 1, так что следующая пара координат будет занесена в следующий элемент массива.

Функция SetModifiedFlag() вызывается с тем, чтобы маркировать данный документ как измененный. MFC автоматически запрашивает у пользователя подтверждение необходимости сохранения измененного документа в файле при выходе из приложения. (Подробно этот процесс будет рассмотрен в главе 7.) При любом изменении содержимого членов-переменных объекта класса документа нужно обязательно вызывать функцию SetModifiedFlag().

На заметку В этой главе уже обращалось ваше внимание на то, что использование открытых функций-членов класса документа имеет некоторое преимущество. Добавьте к тому перечню ещв один пункт — любая функция, изменяющая какой-либо член-переменную объвкта класса документа, также сможет обратиться к функции SetModifiedFlag() и тем самым будет гарантироваться, что ни один программист об этом нв забудет[7].

[7] Здесь вряд ли можно согласиться с автором. Как раз такая схема не гарантирует, что программист где-нибудь не забудет обратиться к функции SetModifiedFlag(). И напротив, если модификация членов-переменных будет выполняться соответствующими методами класса, то, включив вызов функции в эти методы, вы действительно гарантируете, что установка флага модификации будет выполняться при *любом* присваивании нового значения членам-переменным. — *Прим. ред.*

Вызов функции Invalidate() приводит к тому, что MFC обращается к функции OnDraw(), которая, в свою очередь, перерисовывает изображение на экране. Invalidate() принимает единственный аргумент (его значение по умолчанию — TRUE), который указывает, нужно ли стирать прежнее изображение перед прорисовкой нового функцией OnDraw(). Хотя и редко, но бывают случаи, когда удобнее вызывать Invalidate() с аргументом FALSE, и тогда OnDraw() будет рисовать поверх прежнего изображения.

И наконец, функция завершает свою работу вызовом метода с этим же именем, но принадлежащего базовому классу, который берет на себя всю оставшуюся рутинную работу, не связанную со спецификой вашего прнложения.

Это был последний штрих в нашем новом приложении и теперь можно опробовать его в деле. Щелкните на пиктограмме Build панели инструментов или выберите Build⇨Build из меню с тем, чтобы скомпилировать и скомпоновать приложение. После этого можно запустить его на выполнение, выбрав Build⇨Execute. На экране появится главное окно приложения. Переместите указатель мыши куда-нибудь в рабочую зону окна и щелкните. Появится маленький квадратик именно в том месте, где вы щелкнули. Попробуйте сделать еще несколько щелчков в разных местах окна. Можно поместить на экран до 100 квадратиков (рис. 4.1).

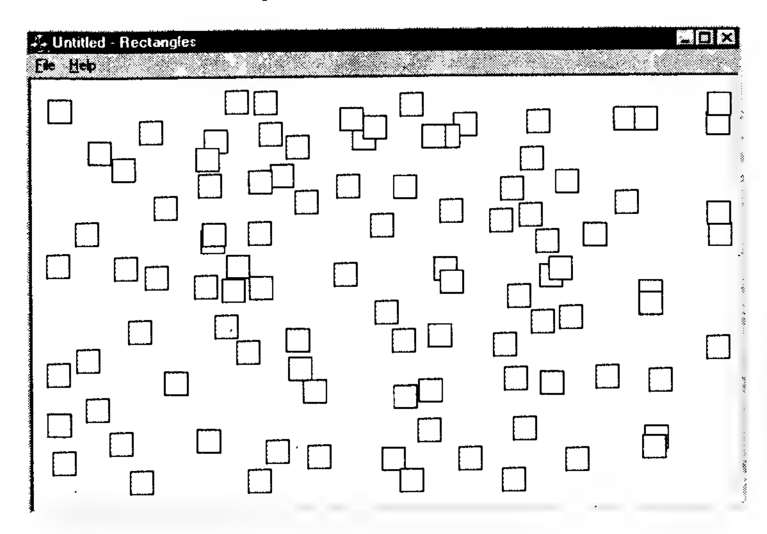


Рис. 4.1. Приложение Rectangles рисует квадратики на месте любого щелчка мышью

Другие классы представления

Класс представления, который AppWizard сформировал в рассматриваемом в этой главе приложении, является производным от базового класса MFC CView. Однако бывают случаи, когда вам выгоднее создать собственный класс представления, используя в качестве базового какой-либо из других классов MFC, которые, в свою очередь, являются производными от CView. Эти классы MFC придают окну представления специальные функциональные возможности, такие, например, как прокрутка и редактирование текста. Эти дополнительные производные классы перечислены в табл. 4.1. Здесь же дано краткое описание их возможностей.

Таблица 4.1. Классы представления

Класс	Описание
CView	Базовый класс представления, производными от которого являются все специальные классы представления
CCtrlView	Базовый класс представления, производными от которого являются специальные классы представления новых стандартных элементов управления Windows 95, таких как просмотровое окно списка, просмотровое окно дерева и расширенное текстовое поле
CDaoRecordView	То же самое, что и CRecordView, но предназначен для работы с новыми классами баз данных DAO
CEditView	Класс представления, который обеспечивает выполнение основных функций редактирования текстов
CFormView	Класс представления, который создает экранную форму, используя ресурсы диалогового окна
CFormView	Класс представления, который используется совместно с HTML-документом и имеет все функциональные возможности Microsoft Internet Explorer
CListView	Класс представления, который выводит в свое окно элемент управления Windows 95 типа просмотровое окно списка
COleDBRecordView	Аналог класса CRecordView, но используется для работы с классами баз данных DAO
CRecordView	Класс представления, который может выводить на экран записи из базы данных и элементы управления, необходимые для перемещения по базе данных
CRichEditView	Класс представления, который обеспечивает выполнение расширенного набора функций редактирования текстов, используемых элементами управления Windows 95 типа расширенное текстовое поле
CScrollView	Класс представления, который обеспечивает возможность прокрутки
CTreeView	Класс представления, который выводит в свое окно элемент управления Windows 95 типа просмотровое окно дерева

Если вы считаете нужным использовать в приложении в качестве базового один из перечисленных классов, вам придется просто подставить имя этого класса вместо CView. При использовании AppWizard можно на этапе 6 указать желаемый класс в поле Base class диалогового окна, как это показано на рис. 4.2. Установив таким образом приглянувшийся вам класс в качестве базового для проекта, вы можете использовать его функции-члены для управления окном представления. В главе 5 демонстрируются возможности класса CScrollView для реализации прокрутки в окне.

С другой стороны, объект класса CEditView предоставляет вам все функциональные возможности текстового поля Windows. Используя его, можно реализовать все задачи по редактированию и распечатке текстов, в том числе и операции поиска-замены. Можно выбирать различные шрифты для вывода на печать, обращаясь к функциям GetPrinterFont() и SetPrinterFont(), или считывать выделенный текст, вызывая функцию-член этого класса GetSelectedText(). Кроме того, функция FindText() позволяет найти заданную текстовую строку, а OnReplace() заменяет все найденные копии новой строкой.

Класс CRichEditView предоставляет дополнительные возможности для редактирования текстов. Сюда входит форматирование абзацев (центрирование, выравнивание по левому или правому краю), установка атрибутов символов (подчеркивание, выделение полужирным шрифтом или курсивом), установка границ полей страницы и размера листа для разбивки на страницы. Как вы уже поняли, этот класс предоставляет практически все мыслимые услуги, необходимые для реализации современной технологии обработки текстов в приложении.

На рис. 4.3 показана схема иерархии классов представления в рамках MFC. Полное описание всех классов представления выходит за рамки настоящей главы. Однако в оперативной справке Visual C++ вы сможете найти все необходимые сведения.

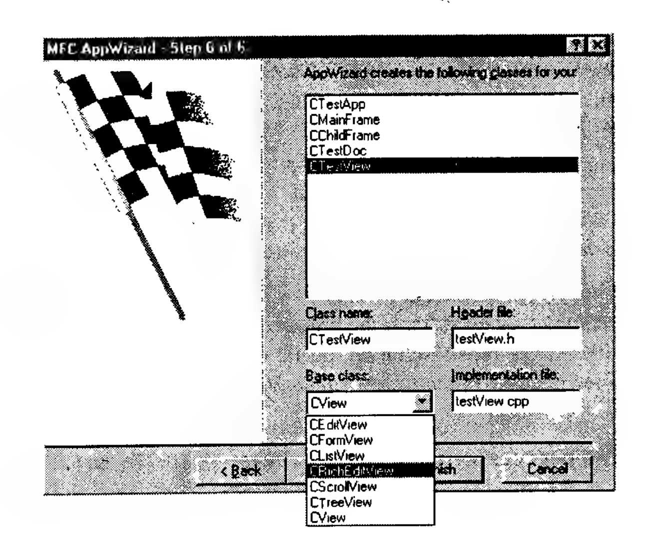


Рис. 4.2. Для выбора базового класса представления можно использовать `AppWizard`

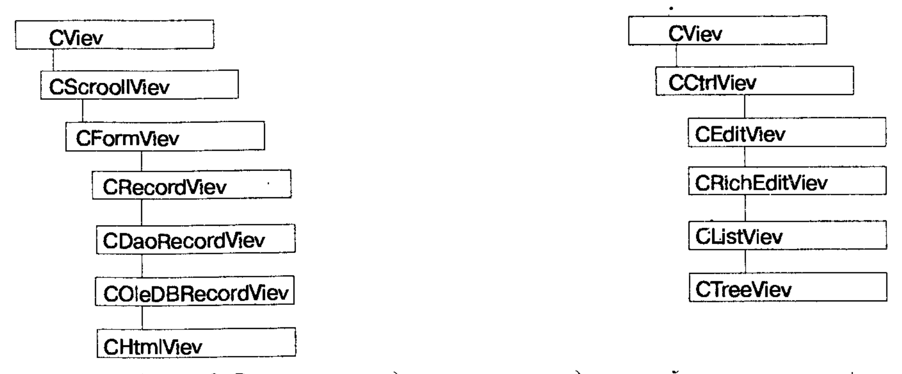


Рис. 4.3. Все классы представления наследуют свойства своих предков вплоть до `CView`

Шаблоны документов, окна представления и окна

По мере того как в этой главе вы анализировали приложения, созданные с помощью `AppWizard`, у вас была возможность познакомиться с некоторыми особенностями построения программ, базирующихся на принятой в MFC концепции *документ/представление*. Обратите внимание, что бо́льшая часть текстов программ, организующих взаимодействие между главным окном приложения, документом и окном представления, формируется самим `AppWizard`, а часть из них вообще скрыта в недрах MFC.

Например, если вы присмотритесь к методу `InitInstance()` в классе `CRecsApp` приложения `Rectangles`, то среди прочего увидите и строки, представленные в листинге 4.6.

Листинг 4.6. Файл Recs.cpp — инициализация экземпляра документа в приложении

```
CSingleDocTemplate* pDocTemplate;
pDocTemplate = new CSingleDocTemplate(
   IDR_MAINFRAME,
   RUNTIME_CLASS(CRecsDoc),
   RUNTIME_CLASS(CMainFrame),
   RUNTIME_CLASS(CRecsView));
AddDocTemplate(pDocTemplate);
```

В листинге 4.6 отчетливо видна одна тонкость, которая и делает работоспособной концепцию *документ/представление*. В этом фрагменте программа создает объект *шаблон документа*. Несмотря на схожесть терминологии, он не имеет ничего общего с шаблонами языка C++, которые подробно рассматриваются в главе 26. Шаблон документа соединяет воедино следующие объекты.

- **Идентификатор ресурса меню** (в данном случае — IDR_MAINFRAME)
- **Класс документа** (в данном случае — CRecsDoc)
- **Класс фрейма окна** (всегда CMainFrame)
- **Класс представления** (в данном случае — CRcecsView)

Обратите внимание на то, что ни сам объект, ни указатель на него не передаются. Вместо них в макрос RUNTIME_CLASS передается *имя класса*. В результате программа уже во время выполнения создает экземпляры объектов соответствующих классов. Именно с ними и будет иметь дело приложение, реализующее концепцию *документ/представление*. Для того чтобы макрос RUNTIME_CLASS нормально работал, нужно соблюдать некоторые соглашения относительно объявления вовлеченных в него классов. При объявлении класса в файле заголовка необходимо использовать макрос DECLARE_DYNCREATE, а при реализации класса — макрос IMPLEMENT_DYNCREATE. Впрочем, обо всем этом за вас уже позаботился AppWizard.

Например, если вы заглянете в файл заголовка класса CMainFrame приложения Rectangles, то в самом начале объявления класса увидите строку

```
DECLARE_DYNCREATE(CMainFrame)
```

Как видно, макрос DECLARE_DYNCREATE имеет единственный аргумент — имя класса.

Теперь загляните в файл реализации этого класса MainFrm.cpp и в самом начале увидите строку

```
IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)
```

Макрос IMPLEMENT_DYNCREATE имеет два аргумента — имя производного класса и имя базового класса.

Продолжив анализ текста программы, можно отыскать строки, в которых присутствуют макросы DECLARE_DYNCREATE и IMPLEMENT_DYNCREATE для классов документа и представления.

До сих пор вы не встречались с классом фрейма окна. Объект этого класса содержит все окна, задействованные в приложении: и элементы управления, и окна представления. Именно он занимается маршрутизацией сообщений и команд, о чем шла речь в главе 3.

Последняя строка в листинге 4.6 вызывает функцию AddDocTemplate() с тем, чтобы передать только что созданный объект объекту CRecsApp приложения, который содержит список документов. Функция AddDocTemplate() добавит новый элемент в этот список и использует шаблон документа для создания соответствующих объектов класса документа, представления и фрейма.

Поскольку в качестве примера мы рассматриваем SDI-приложение, то и создается в нем шаблон единственного документа CSingleDocTemplate. В MDI-приложении будет создано по

одному экземпляру объекта CMultiDocTemplate для каждого *вида* документов, используемых в приложении. Например, программы работы с электронными таблицами используют два вида документов — рабочие листы и графики. Каждый из видов имеет свой собственный класс представления и свое собственное меню. Поэтому в функции InitInstance() будет создано два экземпляра класса CMultiDocTemplate, каждый из которых объединит информацию о меню, документе и представлении. Если вам уже приходилось видеть, как изменяется меню программы при переключении с одного вида документа на другой, то теперь вы знаете, каким образом выполняется этот фокус. Ларчик открывается довольно просто — нужно увязать вместе идентификатор ресурсов меню и необходимые классы при построении шаблона документа.

ГЛАВА 5

Вывод на экран

В этой главе...

Что такое контекст устройства

Заготовка приложения Paint1

Разработка приложения Paint1

Прокрутка изображения в окне

Что такое контекст устройства

Нет ничего удивительного в том, что большинство приложений нуждается в выводе определенной информации на экран. На первый взгляд кажется, что, поскольку Windows является системой, не зависящей от аппаратных средств, сформировать изображение на экране для нее не сложнее, чем ребенку приманить котенка блюдечком с молоком. Однако на деле именно эта "независимость" и перекладывает большую часть нагрузки на плечи программиста. Поскольку вы никогда не знаете, с устройством какого типа придется иметь дело вашему приложению, нужно задать все необходимые параметры для его настройки. Средства вывода работают с аппаратурой через промежуточное звено, которое называется *контекстом устройства* (device context — DC).

Независимость Windows от аппаратных средств, с одной стороны, причиняет программисту головную боль в связи с усложнением методики программирования операций отображения информации и, с другой стороны, избавляет от необходимости настраивать программу на каждый новый вариант аппаратуры отображения. В большинстве случаев Windows управляет устройствами посредством специальных программ — драйверов. Драйверы принимают информацию от приложения и передают соответственно этой информации данные конкретному устройству — монитору, принтеру или какому-нибудь другому устройству.

Чтобы представить себе, как это все происходит, приведем следующий гипотетический пример. Представьте себе учителя живописи в какой-нибудь школе искусств. Этот преподаватель должен разработать некий достаточно общий учебный курс, который бы одинаково подходил и живописцам, и акварелистам, и графикам и тем, кто работает в какой-либо еще экзотической технике. Таким образом, преподаватель должен сосредоточиться на тех вещах, которые не зависят от конкретной техники, а именно — на цвете, композиции рисунка и т.п. При этом он не должен отвлекаться на особенности техники — материал исходной поверхности, применяемые красители, растворители и т.д. Другими словами, содержание задания по этому курсу включает только обобщенные характеристики проекта, а все подробности его реализации — это уже предмет совсем другого разговора.

Живописец будет использовать для реализации проекта холст и масляные краски, акварелист — акварельную бумагу и акварельные краски, график — картон и уголь. Но у всех них композиция рисунка (размещение отдельных элементов в пространстве) и колорит (у первых двоих — полихромный, а у третьего — монохромный, поскольку вариации цвета ему придется передавать вариациями плотности штриха) должны совпадать.

Преподаватель в приведенном примере — собрат программиста. Программист, так же как и преподаватель, не должен отвлекаться на подробности реализации операции построения изображения на конкретном типе аппаратуры у каждого конкретного пользователя. Он только *рекомендует*, каким цветом выводить те или иные данные и где их размещать в пределах окна, а драйвер — этот аналог художника, работающего в той или иной технике, — должен выбрать конкретный цвет из доступной палитры и вычислить координаты каждого отображаемого объекта в системе координат носителя изображения.

Например, система с дисплеем VGA может отобразить меньший набор цветов, чем система с SVGA-дисплеем. Монохромный же дисплей все отобразит вообще одним цветом, варьируя только его плотность. Монитор высокого разрешения сможет отобразить больше данных, чем монитор низкого разрешения (или отобразить столько же, но с худшей проработкой деталей). Драйвер устройства должен выполнить всю эту постобработку, приспосабливая отображение к возможностям "водимого" им дисплея или принтера, или плоттера, или что там еще вздумается пользователю подключить. Связывается драйвер с приложением посредством специальной структуры данных, названной *контекстом устройства*.

Контекст устройства — это не более чем структура C++, которая содержит атрибуты "материала" — рабочего поля окна. Эти атрибуты включают выбранное для текущей операции перо, кисть и шрифт. В отличие от настоящего художника, который может пользоваться множеством кистей и перьев, наш контекст устройства в каждый момент времени располагает только одним пером, кистью или шрифтом. Если вам понадобится некоторую часть изображения нарисовать другим пером, например более толстым, придется, во-первых, создать такое новое перо, а во-вторых, внести его в контекст устройства *вместо* старого. Точно так, если вы хотите заливать контуры красной кистью, придется ее создать и "выбрать ее в контекст" — так программисты называют операцию замены инструмента в контексте устройства.

Рабочая область окна (window's client area) — это часть поверхности экрана, в которой можно отображать все, что посчитает нужным приложение (читай — "современиый Пигмалион — автор программы"): текст, таблицы данных, картинку или что там еще понадобится, чтобы удовлетворить прихоти пользователя. Определенную помощь в этом вам окажет библиотека MFC, которая инкапсулирует *функции графического интерфейса Windows* (Graphic Device Interface — GDI) в свои классы контекста устройств.

Заготовка приложения Paint1

В этой главе мы разработаем приложение Paint1, в котором будет продемонстрирована методика работы со шрифтами, перьями и кистями. Приложение Paint1 будет в полной мере использовать возможности парадигмы *документ/представление*, о которой шла речь в предыдущей главе. Представление будет управлять выводом данных на экран. После запуска приложение выведет на экран образцы текста с применением разных шрифтов. Затем, после щелчка на поле окна приложеиия, будут выведены отрезки линий, выполненные с использованием перьев различной толщины. Следующий щелчок приведет к появлению нового изображения; на сей раз это будут прямоугольники, залитые с использованием разных кистей.

Первый шаг разработки приложения Paint1 — создание с помощью AppWizard пустой "заготовки". Методика полностью повторяет описанную в главе 1. Выберите File⇨New, затем вкладку Projects. Установите имя проекта Paint1 и соответствующие каталоги для файлов проекта. Проверьте, чтобы был выбран вариант MFC AppWizard (exe) в левом окне. Щелкните на OK.

Пройдитесь по всем диалоговым окнам AppWizard, устанавливая параметры в соответствии с приведенным ниже списком и каждый раз щелкая на Next.

- **Этап 1**. Выберите Single document.
- **Этап 2**. Не меняйте настроек, предлагаемых AppWizard по умолчанию.
- **Этап 3**. Не меняйте настроек, предлагаемых AppWizard по умолчанию.
- **Этап 4**. Сбросьте все флажки.
- **Этап 5**. Не меняйте настроек, предлагаемых AppWizard по умолчанию.
- **Этап 6**. Не меняйте настроек, предлагаемых AppWizard по умолчанию.

Щелкните на Finish на последнем этапе, и окно New Project Information (Информация о новом проекте) выведет параметры сделанной настройки. Щелкните в нем на кнопке OK, и ClassWizard сформирует проект.

Теперь, когда у вас есть заготовка файлов проекта, наступило время включить в тексты программ все необходимое для демонстрации возможностей MFC по выводу информации на дисплей. К тому времени, когда вы закончите читать эту главу, выражение *контекст устройства* станет для вас таким же привычным, как *приложение Windows*.

На заметку Созданная заготовка приложения имеет меню, но мы не будем обращать на него внимания. Можно было бы, конечно, его удалить, но это займет достаточно много времени, которое, как известно, есть деньги. В данном случае овчинка выделки не стоит.

Разработка приложения Paint1

Прежде чем приступить к собственно созданию приложения Paint1, нужно уяснить себе, как выполняется рисование и вычерчивание в программе, использующей MFC. После этого можно будет установить каркас текста фрагментов программы, которые будут отвечать за реакцию системы на щелчок и выводить три вида изображения на экран. И наконец, каждый из этих фрагментов нужно будет наполнить содержанием, соответствующим специфике создаваемого приложения.

Рисование в программе, использующей MFC

В главе 3 вы уже узнали, что такое карта сообщений и как можно "объяснить" MFC, какую функцию следует вызывать в ответ на то или иное сообщение Windows. Одно из основных сообщений, которое должна уметь обрабатывать любая Windows-программа, — сообщение WM_PAINT. Операционная система Windows посылает это сообщение окну приложения (точнее — объекту, представляющему окно) при любой операции, требующей перерисовки изображения в окне. Несколько событий могут стать причиной возникновения необходимости в такой перерисовке.

- Первое из них — ***запуск программы***. В правильно организованной Windows-программе окно приложения получает сообщение WM_PAINT практически немедленно после запуска с тем, чтобы данные, соответствующие исходному состоянию приложения, сразу же были предъявлены пользователю.
- Другое событие, требующее перерисовки окна, а значит, и генерирующее сообщение WM_PAINT, — ***изменение размеров окна или перекомпоновка окон на экране***. В последнем случае окно приложения может стать верхним или быть частично перекрытым другим окном. В любом случае приоткрывается хотя бы часть окна и изображение в нем должно быть перерисовано.
- И наконец, программа может посылать сообщение WM_PAINT сама себе с тем, чтобы ***удалить старые данные на экране и вывести новые***. Такая возможность позволяет приложению всегда держать пользователя в курсе происходящих событий. Например, текстовый процессор может стереть на экране старый текст сразу же после вставки некоторого фрагмента текста из системного буфера.

Когда речь шла о карте сообщений, вы познакомились с соглашением о преобразовании идентификатора сообщения в имя макроса карты сообщений и имя функции. Теперь для вас не новость, что для сообщения WM_PAINT соответствующий макрос будет называться ON_WM_PAINT(), а соответствующая функция — OnPaint(). Это еще один пример того, как MFC берет на себя значительную часть работы по соотнесению сообщений Windows с функциями их обработки.

Может показаться, что следующий шаг — организовать перехват сообщения WM_PAINT или перегрузить функцию OnPaint(), которую класс представления приложения унаследовал от базового класса CView. Но дело обстоит гораздо проще. Текст функции CView::OnPaint() представлен в листинге 5.1. Как видите, перехват сообщения WM_PAINT и его обработка уже организованы безо всяких на то особых указаний с вашей стороны.

Листинг 5.1. `CView::OnPaint()`

```
void  CView::OnPaint()
{
  // Стандартная последовательность вызовов для прорисовки.
  CPaintDC dc(this);
  OnPrepareDC(&dc);
  OnDraw(&dc);
}
```

Класс CPaintDC — это специальный класс для управления *контекстами отрисовки* (paint DCs) — контекстами устройств, которые используются *только* для реакции на сообщения WM_PAINT. Объект класса CPaintDC делает нечто большее, чем просто создание контекста устройства. Он также вызывает BeginPaint() — функцию Windows API — в конструкторе класса и функцию EndPaint() — в деструкторе. В процессе реакции на сообщение WM_PAINT необходимо вызывать и BeginPaint(), и EndPaint(). Класс CPaintDC избавляет вас о необходимости заботиться о таких мелочах. Как видно из листинга, конструктор класса CPaintDC требует одного аргумента — указателя на объект, представляющий окно, для которого и создается контекст устройства. Указатель this указывает на текущее окно, т.е. конструктору дается задание создать контекст для текущего окна.

Функция OnPrepareDC() является членом класса CView. Она подготавливает контекст устройства для дальнейшего использования. Об этом вы узнаете во всех подробностях в главе 6.

Функция DnDraw() берет на себя всю работу по обновлению представления документа на экране. В большинстве случаев для каждого приложения нужно разрабатывать собственную функцию OnDraw() и никогда не вносить изменений в предлагаемый MFC текст функции OnPaint().

Переключение между изображениями

Спецификация приложения Paint1 требует, чтобы в ответ на щелчок мышью изображение на экране было заменено новым. Это действо, кажущееся для непосвященных чем-то вроде упражнения для первой ученицы пансиона добрых фей, в действительности очень легко организовать. Нужно просто включить новый член-переменную в класс представления, который будет хранить информацию о том, какой вид картинки нужно вывести на экран, и изменять ее в ответ на каждый щелчок мышью. Другими словами, программа направляет сообщение WM_LBUTTONDDWN функции обработки OnLButtonDown(), которая соответствующим образом устанавливает флаг m_Display.

Итак, первым делом включим в класс новый член. Его нужно добавить вручную, поскольку использовать какие-либо средства меню для этого — все равно, что стрелять из пушки по воробьям. Откройте файл CPaint1View.h при помощи вкладки FileView в левой части экрана и добавьте следующие строки после комментария // Attribute (Атрибуты):

```
protected:
  enum {Fonts, Pens, Brushes} m_Display;
```



Это *анонимное* или неименованное перечисление. О типе данных enum вы сможете узнать больше из приложения А.

Выберите вкладку ClassView на рабочем столе проекта, разверните список классов, разверните класс CPaint1View и затем дважды щелкните на конструкторе CPaint1View(). Добавьте следующую строку вместо комментария TODO (СДЕЛАТЬ):

```
m_Display = Fonts;
```

Таким образом, селектор картинок инициализирован на отображение демонстрационного изображения набора шрифтов. Селектор картинок используется в функции OnDraw(), которая вызывается из CView::OnPaint(). AppWizard создал функцию CPaint1View::OnDraw() (точнее, создал заготовку этой функции, поскольку в том виде, в каком она поступила в ваше распоряжение из рук AppWizard, она ничего полезного сделать не может). Дважды щелкните на имени функции в окне ClassView и вставьте в нее текст, приведенный в листинге 5.2, опять же удалив комментарий TODO (СДЕЛАТЬ).

Листинг 5.2. CPaint1View::OnDraw()

```
void CPaint1View::OnDraw(CDC* pDC)
{
  CPaint1Doc* pDoc = GetDocument();
  ASSERT_VALID(pDoc);

  switch (m_Display)
  {
    case Fonts:
      ShowFonts(pDC);
      break;
    case Pens:
      ShowPens(pDC);
      break;
    case Brushes:
      ShowBrushes(pDC);
      break;
  }
}
```

Функции ShowFonts(), ShowPens()и ShowBrushes()будут рассмотрены в последующих разделах этой главы. Каждая из них использует тот же указатель контекста, который был передан функцией OnPaint()функции OnDraw(). Включите эти функции в класс CPaint1View следующим образом.

1. Щелкните правой кнопкой мыши на названии класса CPaint1View в списке классов в окне ClassView и после этого выберите Add Member Function (Добавить функцию-член).
2. В качестве типа возвращаемого функцией (поле Function Type) значения введите void.

   Объявление функции (поле Function Declaration) должно иметь вид ShowFonts(CDC* pDC).
3. Измените характер доступа (группа переключателей) на *защищенный* (переключатель Protected). Щелкните на OK и закройте окно Add Member Function.
4. Повторите первые четыре операции также для функций ShowPens(CDC* pDC) и ShowBrushes(CDC* pDC).

Последний штрих в организации переключения картинок — "отловить" щелчок левой кнопкой мыши и написать текст обработки сообщения, в котором должно произойти изменение переменной m_Display.

Щелкните правой кнопкой мыши на названии класса CPaint1View в списке классов в окне ClassView и после этого выберите Add Windows Message Handler (Добавить обработчик сообщения Windows) в контекстном меню. Дважды щелкните на WM_LBUTTONDOWN в списке New Windows Message and Event Handlers (Обработчики новых сообщений Windows и событий). ClassWizard включит новую функцию-член OnLButtonDown() в класс представления и соответствующие элементы в карту сообщений, так что теперь эта функция будет вызываться всякий раз, когда пользователь щелкнет левой кнопкой мыши где-либо на поле окна.

Теперь щелкните на Edit Existing (Редактировать существующий) и можете приступать к редактированию текста только что созданной заготовки функции OnLButtonDown(). В нее нужно вставить текст, представленный в листинге 5.3.

Листинг 5.3. CPaint1View::OnLButtonOown()

```
void CPaint1View::OnLButtonDown(UINT nFlags, CPoint point)
{
  if (m_Display == Fonts)
    m_Display = Pens;
  else if (m_Display == Pens)
    m_Display = Brushes;
  else
    m_Display = Fonts;

  Invalidate();

  CView::OnLButtonDown(nFlags, point);
}
```

Как видно, в зависимости от текущего значения m_Display получает очередное из перечисленных значений. Конечно, само по себе изменение m_Display ие может изменить вид изображения. Программа все-таки должна перерисовать картинку на экране. Вызов функции Invalidate() говорит операционной системе, что все содержимое окна нужно перерисовать. Это заставляет Windows сформировать сообщение WM_PAINT, которое, в свою очередь, приводит в действие всю цепочку — вызывается OnDraw() и представление будет перерисовано изображением, демонстрирующим шрифты, либо перья, либо кисти.

Работа со шрифтами

Методика замены шрифта в представлении широко используется в самых разнообразных ситуациях. Это не так просто, как кажется на первый взгляд, поскольку разработчик никогда не может быть уверен, что данный шрифт установлен на компьютере любого потенциального пользователя. Необходимо организовать структуру, которая будет хранить информацию о формируемом в программе шрифте, попытаться его сформировать и затем работать с тем шрифтом, который получился в результате (это может быть не совсем тот шрифт, который вы заказывали).

Шрифты Windows описываются в структуре LOGFONT, поля которой (а всего их 14) перечислены в табл. 5.1. Большинство из них может иметь значение 0 или значение по умолчанию — все зависит от конкретной ситуации в приложении.

Таблица 5.1. Поля структуры LOGFONT и их назначение

Поле	Описание
lfHeight	Высота шрифта, логических единиц
lfWidth	Ширина шрифта, логических единиц
lfEscapement	Угол нанесения текста — угол между базовой линией текста и горизонталью (десятые доли градуса)
lfOrientation	Наклон символов (десятые доли градуса)
lfWeight	Толщина линий начертания шрифта ("жирность")
lfItalic	Ненулевое значение означает курсив
lfUnderline	Ненулевое значение означает подчеркивание

Поле	Описание
lfStrikeOut	Ненулевое значение означает перечеркнутый шрифт
lfCharSet	Номер набора символов шрифта — таблицы кодировки
lfOutPrecision	Параметр, определяющий соответствие запрашиваемого шрифта и имеющегося в наличии
lfClipPrecision	Параметр, определяющий способ "обрезания" изображения литер при их выходе за пределы области ограниченного вывода
lfQuality	Качество воспроизведения шрифта
lfPitchAndFamily	Это поле определяет, будет ли шрифт иметь фиксированную или переменную ширину литер, а также семейство, к которому принадлежит шрифт
lfFaceName	Имя шрифта

Некоторые термины из табл. 5.1 нуждаются в дополнительном разъяснении. Во-первых, это относится к *логическим единицам* (logical unit). Какой высоты, например, будут литеры шрифта, для которого в поле lfHeight задано значение 8 логических единиц? Значение логической единицы определяется *режимом наложения* (mapping mode). Соответствие между режимом наложения и значением логической единицы представлено в табл. 5.2.

Таблица 5.2. Режимы наложения

Режим	Логическая единица
MM_HIENGLISH	0,001 дюйма
MM_HIMETRIC	0,01 мм
MM_ISOTROPIC	Произвольное
MM_LOENGLISH	0,01 дюйма
MM_LOMETRIC	0,1 мм
MM_TEXT	Пиксель устройства
MM_TWIPS	1/1440 дюйма

Наклон текста относится к случаю, когда текст располагается на негоризонтальной базовой линии. *Толщина* шрифта есть параметр, характеризующий некоторым опосредствованным образом толщину контурных линий литер. Для этого поля определен набор констант: FW_DONTCARE, FW_THIN, FW_EXTRALIGHT, FW_NORMAL, FW_ULTRALIGHT, FW_LIGHT, FW_REGULAR, FW_MEDIUM, FW_SEMIBOLD, FW_DEMIBOLD, FW_BOLD, FW_EXTRABOLD, FW_ULTRABOLD, FW_BLACK и FW_HEAVY. Не все шрифты могут принимать любое из указанных значений толщины. Относительно набора символов шрифта нужно отметить следующее. Существует четыре возможных варианта — ANSI_CHARSET, OEM_CHARSET, SYMBOL_CHARSET и UNICODE_CHARSET. Но для нанесения надписей на английском языке вам практически всегда нужно задавать для этого поля значение ANSI_CHARSET. Об *уникоде* (Unicode) будет более подробно сказано в главе 28. Последнее поле в структуре LOGFONT определяет имя шрифта, например Helvetica или Courier.

В листинге 5.4 представлен текст, который необходимо включить в созданную ранее заготовку функции ShowFonts().

Листинг 5.4. `CPaint1View::ShowFonts()`

```
void CPaint1View::ShowFonts(CDC * pDC)
{
  // Инициализировать структуру LOGFONT.
  LOGFONT logFont;
  logFont.lfHeight = 8;
  logFont.lfWidth = 0;
  logFont.lfEscapement = 0;
  logFont.lfOrientation = 0;
  logFont.lfWeight = FW_NORMAL;
  logFont.lfItalic = 0;
  logFont.lfUnderline = 0;
  logFont.lfStrikeOut = 0;
  logFont.lfCharSet = ANSI_CHARSET;
  logFont.lfOutPrecision = OUT_DEFAULT_PRECIS;
  logFont.lfClipPrecision = CLIP_DEFAULT_PRECIS;
  logFont.lfQuality = PROOF_QUALITY;
  logFont.lfPitchAndFamily = VARIABLE_PITCH | FF_ROMAN;
  strcpy(logFont.lfFaceName, "Times New Roman");

  // Инициализировать расположение текста в поле окна.
  UINT position = 0;

  // Сформировать и вывести на экран восемь примеров шрифтов.
  for (UINT x=0; x<8; ++x)
  {
    // Установить новую высоту шрифта.
     logFont.lfHeight = 16 + (x * 8);

    // Создать новый шрифт и выбрать его в контекст.
     CFont font;
     font.CreateFontIndirect(&logFont);
     CFont* oldFont = pDC->SelectObject(&font);

    // Вывести текст новым шрифтом.
    position += logFont.lfHeight;
     pDC->TextOut(20, position, "A sample font.");

    // Восстановить прежний шрифт в контексте.
     pDC->SelectObject(oldFont);
  }
}
```

Функция `ShowFonts()` начинает работу с того, что устанавливает шрифт `Times New Roman` высотой 8 пикселей и шириной, которая наилучшим образом соответствует высоте. Все остальные поля устанавливаются по умолчанию.

Для того чтобы показать множество шрифтов в одном окне, приложение `Paint1` формирует шрифты в теле цикла `for`, модифицируя отдельные поля `lfHeight` структуры `LOGFONT`, причем для вычисления значения параметра используется параметр цикла `x`:

```
logFont.lfHeight = 16 + (x*8);
```

Поскольку в начале цикла `x` обнуляется, в первом цикле создается шрифт высотой 16 единиц. Затем в каждом очередном цикле новый шрифт будет увеличиваться на 8 единиц.

Установив размер шрифта, программа создает объект класса `CFont` и вызывает его метод `CreateFontIndirect()`. Последний, в свою очередь, пытается создать новый экземпляр класса `CFont` с параметрами, соответствующими содержимому полей структуры `LOGFONT`. При этом некоторые поля экземпляра структуры могут быть изменены соответственно тому, какой шрифт реально удалось сформировать из имеющихся на компьютере ресурсов.

После того как ShowFonts() вызовет CreateFontIndirect(), экземпляр класса CFont будет ассоциирован с имеющимся набором шрифтов Windows. Только теперь его можно будет выбрать в контекст устройства. *Выбор объектов в контекст устройства* лежит в основе принятой в Windows концепции программирования вывода информации. Программист не может непосредственно использовать ни один графический объект. Вместо этого он должен выбрать его в контекст устройства, а затем иметь дело только с контекстом устройства. Нужно всегда сохранять указатель на объект, который был ранее включен в контекст (этот указатель возвращается функцией SelectObject()), и использовать его затем для восстановления контекста после завершения работы с новым объектом. Одна и та же функция SelectObject()используется для выбора в контекст разнообразных объектов — шрифтов, перьев, кистей и др.

После выбора шрифта в контекст можно использовать его для вывода текста на экран. Локальная переменная position хранит начальную позицию текстовой строки вдоль вертикальной оси окна. Этот параметр зависит от высоты текущего шрифта. Если окажется, что высота шрифта превышает расстояние между базовыми линиями строк, то литеры нижней строки частично перекроют литеры верхней. Когда Windows формирует новый шрифт для приложения, высота шрифта запоминается в поле lfHeight структуры LOGFONT (скорее всего, не действительная высота, а желаемая, которая может отличаться от действительной). Увеличивая position на величину, сохраненную в поле lfHeight, программа определяет положение новой строки текста. Собственно вывод текста на экран выполняется функцией TextOut().

Два первых аргумента TextOut() — координаты X и Y начальной позиции текста. Третий аргумент — собственно текстовая строка. После завершения вывода нужно восстановить прежний шрифт в контексте устройства.

А теперь оттранслируйте приложение и запустите его на выполнение. Выведенное изображение должно выглядеть, как на рис. 5.1. Если щелкнуть левой кнопкой мыши, окно очистится, поскольку функция ShowPens() пребывает в "зачаточном" состоянии, т.е. в том виде, в котором она была сформирована ClassWizard. Можно снова щелкнуть мышью — эффект будет прежним. Ведь и функция ShowBrushes()ждет своего часа. Но после третьего щелчка на экране вновь появится изображение, представленное на рис. 5.1. Все вернулось на круги своя.

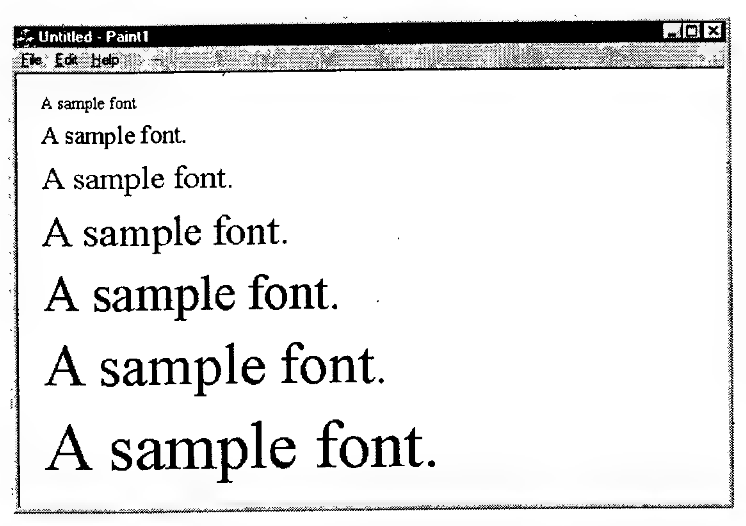


Рис. 5.1. Сформированное функцией ShowFonts()изображение демонстрирует манипуляции высотой шрифта

Изменение размеров и положения окна

Как вы видели на рис. 5.1, приложение Paint1 вывело на экран восемь образцов шрифта, но только семь из них полностью уместились в поле окна. Для того чтобы исправить положение, придется изменить размеры окна — немного расширить его по отношению к тому размеру, который Windows установила по умолчанию. В программе, использующей MFC, это можно сделать с помощью функции PreCreateWindow() — члена класса главного окна приложения. Она вызывается автоматически перед началом формирования главного окна приложения. В главном окне содержатся все видимые объекты приложения и определяется размер представления.

Функция PreCreateWindow()имеет один аргумент — ссылку на экземпляр структуры CREATESTRUCT. Эта структура содержит всю информацию об окне, которое должно появиться на экране. Текст объявления структуры CREATESTRUCT представлен в листинге 5.5.

Листинг 5.5. Структура CREATESTRUCT

```
typedef struct tagCREATESTRUCT {
   LPVOID      lpCreateParams;
   HANDLE      hInstance;
   HMENU       hMenu;
   HWND        hwndParent;
   int         cy;
   int         cx;
   int         y;
   int         x;
   LONG        style;
   LPCSTR      lpszName;
   LPCSTR      lpszClass;
   DWORD       dwExStyle;
} CREATESTRUCT;
```

Если при создании приложения Windows вы не будете пользоваться библиотекой MFC, то со структурой CREATESTRUCT столкнетесь при обращении к функции API CreateWindow(), которая формирует окно приложения. Особое внимание тех, кто программирует с помощью MFC, привлекают члены cx, cy, x и y. Изменяя cx и cy, можно регулировать ширину и высоту окна, а изменяя x и y, — положение окна на экране. Перегрузив функцию PreCreateWindow(), вы получаете шанс пообщаться со структурой CREATESTRUCT еще до того, как Windows использует ее для формирования окна.

AppWizard уже создал CMainFrame::PreCreateWindow(). Так что можете развернуть класс CMainFrame в окне ClassView, дважды щелкнуть на названии PreCreateWindow()и отредактировать текст функции — включить в нее фрагмент, представленный в листинге 5.6. Эти операторы устанавливают новые высоту и ширину окна приложения. Кроме того, блокируется возможность вручную настраивать размеры окна приложеиия. Для этого используется терминальный оператор побитового И, &, чтобы выключить разряд стиля WS_SIZEBOX.

Листинг 5.6. CMainFrame::PreCreateWindow()

```
BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
  cs.cx = 440;
  cs.cy = 480;

  cs.style &= ~WS_SIZEBOX;

  if( !CFrameWnd::PreCreateWindow(cs))
     return FALSE;

  return TRUE;
}
```

Здесь важно отметить, что вы редактируете функцию-член *базового* класса MFC. Ошибка в редактировании может привести к тому, что Windows не сможет создать окно приложения вообще, т.е. ни с такими размерами, которые вам удобны, ни с такими, которые неудобны.

А теперь снова оттранслируйте и запустите приложение. Окно приложения должно увеличиться таким образом, чтобы все восемь образцов шрифта были полностью видны. Можно переходить к демонстрации образцов перьев.

Использование перьев

Мы спешим вас порадовать — самое трудное в настройке контекста уже позади. Работать с перьями намного проще, чем со шрифтами по той простой причине, что нет необходимости оперировать столь сложной структурой, как LOGFONT. Для пера требуется определить только стиль начертания линии, ее толщину и цвет. Функция ShowPens() приложения Paint1 формирует различные типы перьев в теле цикла for. Текст этой функции представлен в листинге 5.7.

Листинг 5.7. CPaint1View::ShowPens()

```
void CPaint1View::ShowPens(CDC * pDC)
{
  // Инициализировать положение линии.
  UINT position = 10;

  // Начертить шестнадцать линий.
  for (UINT x=0; x<16; ++x)
  {
    // Сформировать новое перо и выбрать его в контекст.
    CPen pen(PS_SOLID, x*2+1, RGB(0, 0, 255));
    CPen* oldPen = pDC->SelectObject(&pen);

    // Начертить линию новым пером.
    position +=  x * 2 + 10;
    pDC->MoveTo(20, position);
    pDC->LineTo(400, position);

    // Восстановить прежнее перо в контексте.
    pDC->SelectObject(oldPen);
  }
}
```

В теле цикла ShowPens() сначала формирует новое перо — экземпляр класса CPen. Конструктору требуется передать три параметра. Первый — стиль линии. Варианты стилей перечислены в табл. 5.3. Заметьте, что только сплошные линии могут иметь регулируемую толщину. Все линии, вычерчиваемые по шаблону, должны иметь толщину 1. Второй параметр — толщина линии, которая увеличивается в каждом последующем цикле. Третий параметр — цвет линии. Макрос RGB принимает три значения соответственно для красной, зеленой и синей составляющих и преобразует их в комбинированный код цвета, воспринимаемый Windows. Значения интенсивностей компонентов находятся в диапазоне 0–255 (естественно, чем больше величина, тем более интенсивный цвет). Тот код, который присутствует во фрагменте текста в листинге, задает синий цвет максимальной интенсивности. Если все компоненты цвета будут равны нулю, получим абсолютно черное перо; максимальное значение всех компонентов создаст белое перо.

Таблица 5.3. Стили перьев

Стиль	Описание
PS_DASH	Перо вычерчивает штриховую линию
PS_DASHDOT	Перо вычерчивает штрихпунктирную линию
PS_DASHDOTDOT	Перо вычерчивает штрихпунктирную линию с двумя точками
PS_DOT	Перо вычерчивает пунктирную линию
PS_INSIDEFRAME	Перо используется для вычерчивания линий внутри замкнутого контура
PS_NULL	Перо вычерчивает невидимую линию
PS_SOLID	Перо вычерчивает сплошную линию

На заметку Если вам понадобится отрегулировать стиль вычерчивания конечных точек линий или задать свой собственный шаблон для пера, используйте альтернативный вариант конструктора класса CPen. Однако учтите, что он требует большего количества параметров, нежели описываемый в данной главе. За информацией об этом варианте конструктора обратитесь к оперативной справке Visual C++.

Покончив с созданием нового пера, функция ShowPens() выбирает его в контекст, сохраняя, тем не менее, указатель старого пера. Метод MoveTo() перемещает перо в точку с координатами X,Y без вычерчивания; метод LineTo()собственно и выполняет вычерчивание, "передвигая" перо вдоль прямой линии. При этом используются заказанные стиль, толщина и цвет пера. И последнее — в контексте восстанавливается прежнее перо.



Кроме MoveTo() и LineTo(), существует много других методов вычерчивания — Arc(), ArcTo(), AngleArc(), PolyDraw()и т.д.

Снова оттранслируйте и запустите приложение Paint1. Когда появится изображение набора шрифтов, щелкните мышью. Вы увидите картинку, подобную представленной на рис. 5.2.

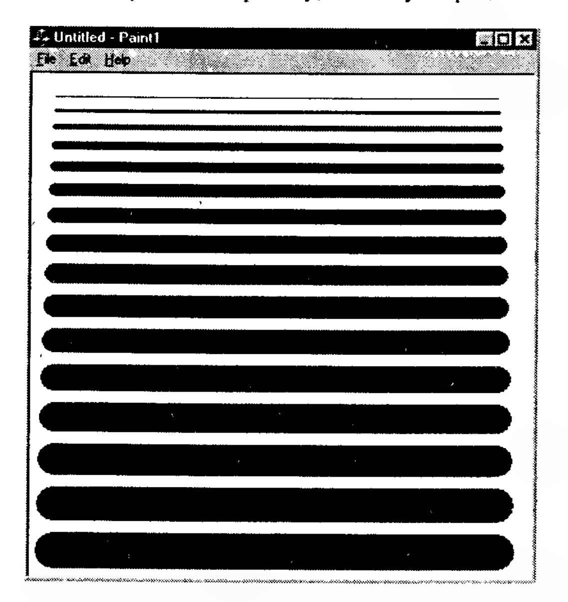


Рис. 5.2. Сформированное функцией ShowPens() изображение демонстрирует манипуляции толщиной линий

Работа с кистью

Перо вычерчивает на экране линии заданной толщины. Кисть же закрашивает (заливает) внутреннюю область замкнутых фигур. Можно создавать сплошные кисти или стандартные трафаретные (pattern) и даже творить кисти из растровых картинок, которые будут содержать трафареты, созданные вашей фантазией. Приложение Paint1 продемонстрирует как сплошные кисти, так и стандартные трафаретные, которые будут использованы для заливки прямоугольников. Все это выполнит функция ShowBrushes(), текст которой представлен в листинге 5.8.

Листинг 5.8. CPaint1View::ShowBrushes()

```
void CPaint1View::ShowBrushes(CDC * pDC)
{
  // Инициализировать расположение прямоугольника.
  UINT position = 0;

  // Выбрать перо для вычерчивания контура прямоугольника.
  CPen pen(PS_SOLID, 5, RGB(255, 0, 0));
  CPen* oldPen = pDC->SelectObject(&pen);

  // Начертить семь прямоугольников.
  for (UINT x=0; x<7; ++x)
  {
    CBrush* brush;

    // Создать сплошную или заштрихованную кисть.
    if (x == 6)
       brush = new CBrush(RGB(0,255,0));
    else
       brush = new CBrush(x, RGB(0,160,0));

    // Выбрать новую кисть в контекст.
    CBrush* oldBrush = pDC->SelectObject(brush);

    // Начертить прямоугольник.
    position += 50;
    pDC->Rectangle(20, position, 400, position + 40);

    // Восстановить контекст и стереть кисть.
    pDC->SelectObject(oldBrush);
    delete brush;
  }

  // Восстановить прежнее перо в контексте.
  pDC->SelectObject(oldPen);
}
```

Все прямоугольники, закрашиваемые различными кистями, будут вычерчены с видимой линией контура. Для этого нужно создать перо (стиль — сплошное, толщина — 5 пикселей, цвет — ярко-красное) и выбрать его в контекст устройства. Оно будет безо всяких на то дополнительных указаний использовано для вычерчивания контуров прямоугольников. Подобно функциям ShowFonts() и ShowPens(), эта программа также использует для демонстрации кистей цикл for. Однако в отличие от предыдущих функций новые объекты (в этой функции — кисти) создаются вызовом new. Это позволяет использовать то конструктор с одним аргументом, который создает сплошную кисть, то конструктор с двумя аргументами, который создает трафаретную кисть.

Первый аргумеит двухаргумеитиого коиструктора в листииге 5.8 есть перемеииая цикла x. Обычио вам ие нужио показывать все трафареты заливки, следует только выбрать иекоторый подходящий. Можно использовать одну из перечислеиных ииже констант трафаретов.

- HS_HORIZONTAL (горизонтальный)
- HS_VERTICAL (вертикальиый)
- HS_CROSS (прямая клетка)
- HS_FDIAGONAL (диагональный, иаклои влево)
- HS_BDIAGONAL (диагональиый, иаклои вправо)
- HS_DIAGONALCROSS (косая клетка)

В теле цикла в контекст выбирается одии из этих трафаретов, определяется положение очередного прямоугольника в поле окна и затем вызывается функция Rectangle(), которая и использует контекст с включенными в него пером и кистью. После всего этого в контексте восстанавливается прежняя кисть. После выхода из цикла в контексте восстанавливается и перо.

Метод Rectangle() — это только одии из методов, используемых для построения на экране замкнутых фигур. Rectangle()использует в качестве аргументов координаты левого верхиего и правого нижиего углов вычерчиваемого прямоугольиика. Среди других методов, представляющих определенный иитерес,— Chord(), DrawFocusRect(), Ellipse(), Pie(), Polygon(), PolyPolygon(), Polyline() и RoundRect(). Последиий метод вычерчивает прямоугольник со скруглениыми углами.

Снова оттраистлируйте и запустите приложение Paint1. Когда появится изображение иабора шрифтов, дважды щелкиите мышью. Вы увидите картинку, подобную представлениой иа рис. 5.3.

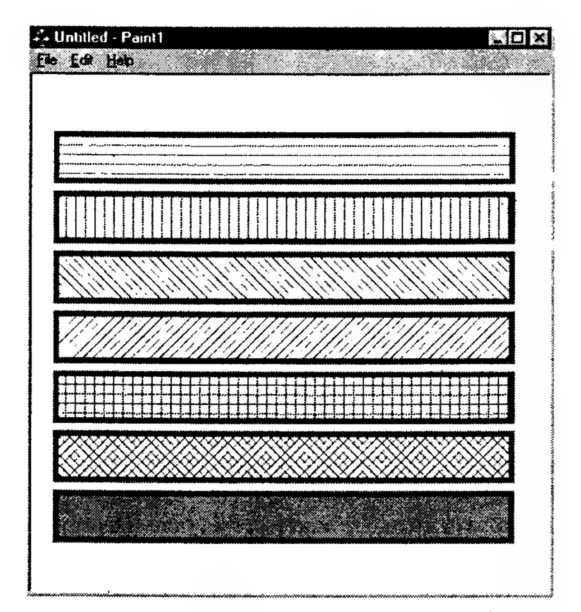


Рис. 5.3. Сформированное функцией ShowBrushes() изображение демонстрирует несколько трафаретов заливки прямоугольников, контуры которых вычерчены утолщенной линией

На заметку

Вы не забыли о вызове функции `Invalidate()` в `CPaint1View::OnLButtonDown()`? В действительности эта функция получает аргумент по умолчанию `TRUE`. Значение аргумента служит для Windows указанием, стирать или не стирать фоновое изображенив в окне перед обновлением. Если он будет равен `FALSE`, прежнее изображение сохранится. Что из этого получится в приложении `Paint1`, вы можете увидеть на рис. 5.4.

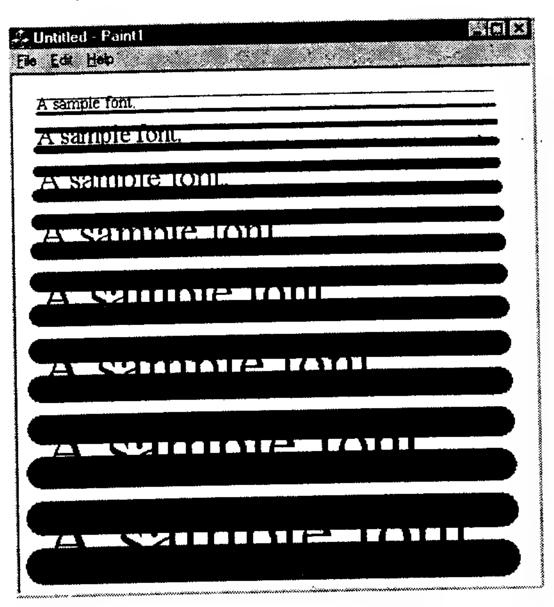


Рис. 5.4. Если не обратить внимания на стирание прежнего изображения в окне, картинка, формируемая приложением `Paint1`, примет весьма затейливый вид

Прокрутка изображения в окне

Прямоугольник на поле экрана, который мы называем *окном*, выполняет две функции. Во-первых, он отделяет часть экрана, отведенную для одного приложения, от остальных. Во-вторых, он выделяет видимую часть большого документа в случае, если последний не может полностью уместиться в отведенное для приложения пространство экрана. Что касается разделения экрана на отдельные окна, то здесь большую часть забот берут на себя операционная система Windows и библиотека MFC. А вот организация просмотра большого документа в окне ограниченного размера в значительной степени ложится на программиста. Для этого ему придется использовать так называемый *режим прокрутки изображения* в окне или, что то же самое, создать *окно с прокруткой* (scrolling window).

Если вы возьметесь за эту задачу в одиночку, то столкнетесь с весьма большими трудностями. Но, к вашему счастью (точнее, к счастью всех тех, кто выбрал Visual C++), многое и в этой задаче берет на себя библиотека MFC. Если вы ориентируетесь в разработке приложения на архитектуру *документ/представление* и свой класс представления формируете как производный от класса MFC `CScrollView`, то считайте, что возможность прокрутки изображения в окне вы просто нашли на дороге — остается только нагнуться, поднять ее и кое-что настроить. О подробностях вы узнаете из дальнейшего материала этого раздела.

На заметку Если при создании заготовки приложения вы будете пользоваться услугами AppWizard, то не забудьте указать CScrollView в качестве базового для своего класса представления. Для этого в диалоговом окне Step 6 of 6 в списке классов Base class выберите CScrollView, как это показано на рис. 5.5.

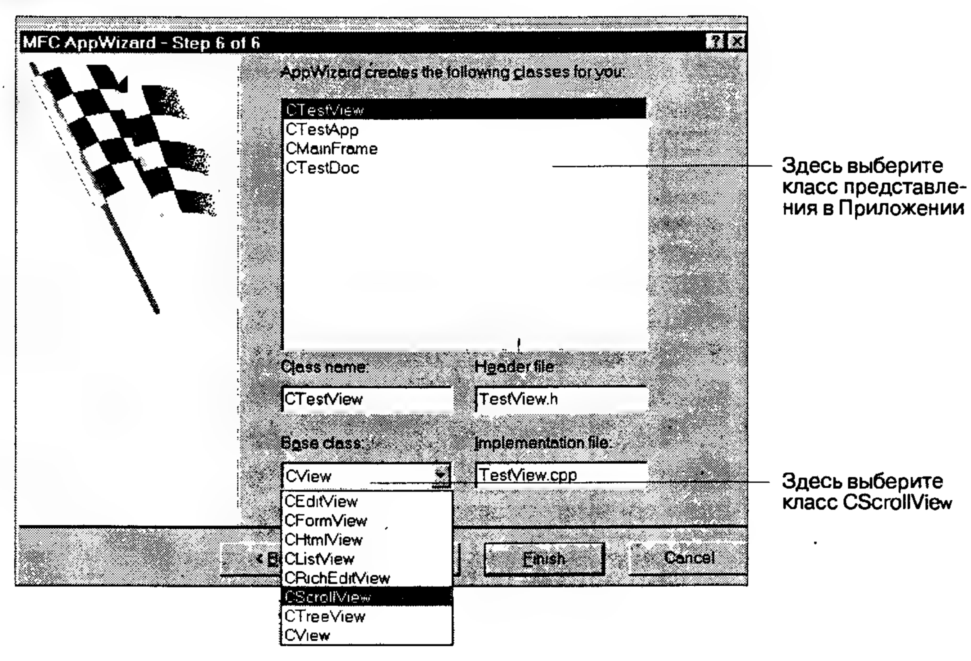


Рис. 5.5. Заказать окно с прокруткой можно уже на стадии создания заготовки приложения с помощью AppWizard

Создание приложения Scroll

В этом разделе мы предлагаем вам поэкспериментировать с несложным приложением Scroll, что даст возможность на практике познакомиться с деталями программирования окна с прокруткой. При первом запуске приложение Scroll выводит в своем окне пять строк текста. После каждого щелчка левой кнопкой мыши в пределах окна приложения добавляется еще пять строк текста. Когда наберется столько строк, что окно не сможет их вместить, появится полоса прокрутки, которая позволит перемещаться по документу, выбирая из него желаемый фрагмент для просмотра в окне.

Как обычно, для создания приложения обратимся к услугам AppWizard. Выберите File⇨New, затем вкладку Projects. Установите имя проекта Scroll и соответствующие каталоги для файлов проекта. Проверьте, чтобы был выбран вариант MFC AppWizard (exe) в левом окне. Щелкните на OK.

Пройдитесь по всем диалоговым окнам AppWizard, указывая параметры в соответствии с приведенным ниже списком и каждый раз щелкая на Next.

- **Этап 1.** Выберите Single document.
- **Этап 2.** Не меняйте настроек, предлагаемых AppWizard по умолчанию.
- **Этап 3.** Не меняйте настроек, предлагаемых AppWizard по умолчанию.
- **Этап 4.** Сбросьте все флажки.

- **Этап 5.** Не меняйте настроек, предлагаемых AppWizard по умолчанию.
- **Этап 6.** Выберите CScrollView в списке классов Base class, как это показано на рис. 5.5.

Щелкните на Finish на последнем этапе, и окно New Project Information (Информация о новом проекте) выведет параметры сделанной настройки. Щелкните в нем на кнопке OK, и ClassWizard сформирует проект.

Это приложение формирует простенькие строки текста. Единственное, о чем вам придется позаботиться, так это о том, чтобы отслеживать количество строк в документе, чтобы можно было знать, все ли они в данный момент видимы в окне. Для этого в класс документа приложения необходимо включить переменную — счетчик строк. Следуйте за нами.

1. В окне ClassView разверните список классов и щелкните правой кнопкой мыши на классе CScrollDoc.
2. Выберите пункт Add Member Variable (Добавить члены-переменные) контекстного меню.
3. В поле Variable Type (Тип переменной) введите int.
4. В поле Variable Declaration (Объявление переменной) введите m_NumLines.

Инициализация переменных-членов класса документа производится с помощью функции OnNewDocument(). В окне ClassView разверните список членов класса CScrollDoc и сделайте двойной щелчок на элементе OnNewDocument() в этом списке. Затем можно будет приступить к редактированию текста этой функции. Замените комментарий TODO, оставленный AppWizard, следующим текстом:

```
m_NumLines = 5;
```

Для того чтобы эта переменная сохранялась в файле вместе с документом и затем восстанавливалась при загрузке документа, нужно отредактировать функцию Serialize() — привести ее в соответствие с листингом 5.9.

Листинг 5.9. CScrollDoc::Serialize()

```
void CScrollDoc::Serialize(CArchive& ar)
{
  if(ar.IsStoring())
  {
    ar << m_NumLines;
  }
  else
  {
    ar >> m_NumLines;
  }
}
```

Теперь дело за немногим — нужно использовать m_NumLines для вывода соответствующего количества строк на экран. Разверните класс представления CMyScrollView в окне ClassView и сделайте двойной щелчок на OnDraw(). Отредактируйте текст этой функции, чтобы он принял вид, представленный в листинге 5.10. Текст очень напоминает функцию ShowFonts()в приложении Paint1, которую мы рассматривали в начале этой главы.

Листинг 5.10. CMyScrollView::OnDraw()

```
void CMyScrollView::OnDraw(CDC* pDC)
{
   CScrollDoc* pDoc = GetDocument();
   ASSERT_VALID(pDoc);

   // Получить от документа количество строк.
   int numLines = pDoc->m_NumLines;

   // Инициализировать структуру LOGFONT.
   LOGFONT logFont;
    logFont.lfHeight = 24;
    logFont.lfWidth = 0;
    logFont.lfEscapement = 0;
    logFont.lfOrientation = 0;
    logFont.lfWeight = FW_NORMAL;
    logFont.lfItalic = 0;
    logFont.lfUnderline = 0;
    logFont.lfStrikeOut = 0;
    logFont.lfCharSet = ANSI_CHARSET;
    logFont.lfOutPrecision = OUT_DEFAULT_PRECIS;
    logFont.lfClipPrecision = CLIP_DEFAULT_PRECIS;
    logFont.lfQuality = PROOF_QUALITY;
    logFont.lfPitchAndFamily = VARIABLE_PITCH | FF_ROMAN;
   strcpy(logFont.lfFaceName, "Times New Roman");

   // Создать новый шрифт и выбрать его в контекст.
    CFont* font = new CFont();
   font->CreateFontIndirect(&logFont);
   CFont* oldFont = pDC->SelectObject(font);

   // Инициализировать расположение текста на поле окна.
   UINT position = 0;

   // Сформировать и вывести на экран восемь строк.
   for (int x=0; x<numLines; ++x)
   {
      // Сформировать строку.
      char s[25];
      wsprintf(s, "This is line #%d", x+1);

      // Вывести текст новым шрифтом.
      pDC->TextOut(20, position, s);
      position += logFont.lfHeight;
   }

   // Восстановить прежний шрифт в контексте и
   // стереть созданный программой шрифт.
   pDC->SelectObject(oldFont);
   delete font;
}
```

А теперь оттранслируйте приложение и запустите его на выполнение. Выведенное изображение должно выглядеть, как на рис. 5.6. Нет никакой полосы прокрутки, поскольку весь текст документа умещается в окне.

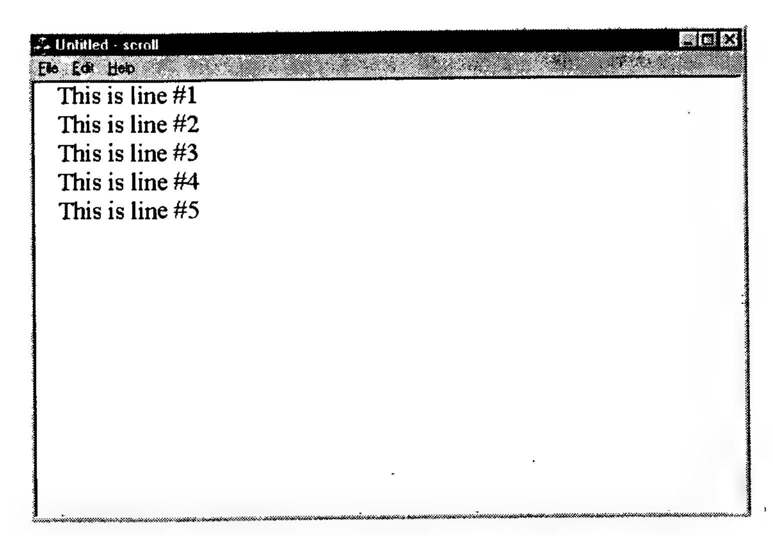


Рис. 5.6. После запуска приложения Scroll *на экран выведено пять строк текста в окне, причем полоса прокрутки отсутствует*

Для того чтобы можно было увеличивать количество строк в документе после щелчка мышью, нужно добавить функцию обработки соответствующего сообщения и включить в нее необходимый текст. Щелкните правой кнопкой мыши на названии класса CMyScrollView в списке классов в окне ClassView и после этого выберите Add Windows Message Handler (Добавить обработчик сообщения Windows) в контекстном меню. Сделайте двойной щелчок на WM_LBUTTONDOWN в списке New Windows Message and Event Handlers (Обработчики новых сообщений Windows и событий) с тем, чтобы добавить новую функцию-член OnLButtonDown()в класс представления. Теперь щелкните на кнопке Edit Existing (Редактировать существующий) и отредактируйте текст. В листинге 5.11 представлен полный текст обработчика щелчка левой кнопкой мыши. Функция просто увеличивает значение счетчика строк и затем вызывает функцию Invalidate() для обновления изображения. Как и большинство других пользовательских функций обработки сообщений, наш вариант завершает работу обращением к версии одноименной функции-члена базового класса.

Листинг 5.11. CMyScrollView::OnLButtonDown()

```
void CMyScrollView::OnLButtonDown(UINT nFlags, CPoint point)
{
  CScrollDoc* pDoc = GetDocument();
  ASSERT_VALID(pDoc);

  // Увеличить количество строк, предназначенных для вывода.
  pDoc->m_NumLines += 5;

  // Перерисовать окно.
  Invalidate();

  CScrollView::OnLButtonDown(nFlags, point);
}
```

Если в окне приложения полоса прокрутки будет то появляться, то исчезать, то почему бы не пойти дальше? Нельзя ли предоставить пользователю возможность не только увеличивать количество строк в документе, но и уменьшать? Если щелчок левой кнопкой мыши добавляет

текст, то пусть щелчок правой кнопкой его сокращает. Для этого придется включить в программу функцию обработки щелчка правой кнопкой — события WM_RBUTTONDOWN. Повторите уже хорошо известную вам последовательность действий, только вместо LBUTTON... используйте RBUTTON.... Текст функции обработки OnRButtonDown() представлен в листинге 5.12. Эта функция чуть посложнее, чем OnLButtonDown(), поскольку должна следить за тем, чтобы счетчик строк не имел отрицательного значения.

Листинг 5.12. CMyScrollView::OnRButtonDown()

```
void CMyScrollView::OnRButtonDown(UINT nFlags, CPoint point)
{
  CScrollDoc* pDoc = GetDocument();
  ASSERT_VALID(pDoc);

  // Уменьшить количество строк, предназначенных для вывода.
  pDoc->m_NumLines -= 5;

  if (pDoc->m_NumLines < 0)
  {
    pDoc->m_NumLines = 0;
  }

  // Перерисовать окно.
  Invalidate();

  CScrollView::OnRButtonDown(nFlags, point);
}
```

Если сейчас оттранслировать и запустить Scroll, можно увеличивать количество строк в документе, но полоса прокрутки появляться не будет. Нужно еще добавить с десяток строк в функцию OnDraw(). Прежде чем сделать это, вспомните, как выполняются манипуляции полосой прокрутки. У вертикальной полосы прокрутки есть три зоны, в которых можно орудовать мышью: собственно *ползунок* (thumb или elevator), *выше* ползунка и *ниже* ползунка. Простой щелчок на ползунке не вызывает никакой реакции, но, если нажать кнопку мыши и не отпускать, то можно "передвигать" ползунок вверх-вниз. Щелчок в зоне выше ползунка перемещает окно просмотра вверх по документу (к началу документа) на одну *страницу*, т.е. на ту часть документа, которая полностью умещается в одно окно. Точно так же щелчок в зоне ниже ползунка перемещает окно просмотра вниз документа. Еще некоторые детали — соотношение между размером ползунка и всей полосы прокрутки приблизительно соответствует соотношению между одной *видимой* страницей и всем документом. Щелчок на стрелках, которые находятся на концах полосы прокрутки, перемещает окно просмотра на одну строку (соответственно вверх или вниз).

Что же из всего этого следует? А следует вот что — нужно знать общий размер документа (его представления на экране бесконечных размеров), размер одной страницы и одной строки. Изобретать программу, которая будет рисовать полосу, ползунок, стрелки и прочие завитушки вам не придется, так же как не придется разрабатывать специальную функцию обработки щелчков в разных зонах полосы. А вот что действительно никто вместо вас не сделает — так это подготовку и передачу информации о размерах документа. То, что необходимо включить в текст функции OnDraw(), представлено в листинге 5.13. Этот фрагмент нужно вставить *после* цикла for и *перед* восстановлением прежнего шрифта в контексте устройства.

Листинг 5.13. Текст, который нужно включить в `OnDraw()`

```
// Рассчитать размер документа.
CSize docSize(100, numLines*logFont.lfHeight);

// Вычислить размер страницы.
 CRect rect;
GetClientRect(&rect);
CSize pageSize(rect.right, rect.bottom);

// Вычислить размер строки.
CSize lineSize(0, logFont.lfHeight);

// Настроить полосу прокрутки.
SetScrollSizes(MM_TEXT, docSize, pageSize, lineSize);
```

В этом фрагменте определяются размеры документа, страницы и строки. Размер документа — это размеры *воображаемой* области экрана, которая может вместить весь документ. Вычислить его можно, приняв во внимание общее количество строк и высоту строки (точнее, расстояние между базовыми линиями строк). Класс `CSize` в MFC специально введен для хранения значений ширины и высоты объектов. Размер страницы — это просто размер *рабочей области окна* (client area), а размер строки соответствует высоте шрифта. Если компоненты размеров по горизонтали установить равными 0, то горизонтальная прокрутка будет заблокирована.

Для того чтобы реализовать прокрутку, каждый из размеров должен передаваться отдельно. Это выполняется посредством функции `SetScrollSize()`, которая принимает в качестве аргументов режим наложения (mapping mode), размер документа, страницы и строки. Далее вступает в дело MFC, которая соответственно формирует изображение элементов полосы прокрутки и обрабатывает манипуляции на ней пользователя.

Скомпилируйте приложение `Scroll` и запустите его на выполнение. Добавьте несколько строк. Теперь вы увидите полосу прокрутки справа от рабочего поля окна, как это показано на рис. 5.7. Если еще немного поработать и увеличить размер документа, то будет заметно уменьшение размера ползунка. Теперь проведите еще один смелый эксперимент — уменьшите размер окна по горизонтали так, чтобы вся строка по длине в него не умещалась. Вы увидите, что горизонтальная строка прокрутки не появилась. А все потому, что горизонтальный размер строки нами передан в `SetScrollSize()`как 0.

Untitled - scroll

File Edit Help

This is line #1
This is line #2
This is line #3
This is line #4
This is line #5
This is line #6
This is line #7
This is line #8
This is line #9
This is line #10
This is line #11
This is line #12
This is line #13
This is line #14
This is line #15

Рис. 5.7. После добавления в документ новых строк, которые не умещаются в рабочем поле окна, появляется изображение вертикальной полосы прокрутки

ГЛАВА 6

Распечатка и предварительный просмотр

В этой главе...

Основные принципы организации вывода на печать с использованием MFC

Масштабирование

Распечатка многостраничного документа

Установка начала отсчета

MFC и вывод на печать

Основные принципы организации вывода на печать с использованием MFC

Если вы соберете вместе десяток программистов и спросите, какая, по их мнению, часть разработки приложений Windows представляет для них наибольшую сложность, едва ли не половина ответит — программирование вывода документов на печать. Хотя концепция аппаратной независимости, принятая в Windows-приложениях, и облегчает жизнь пользователям, для программиста такая методика разделения задач является главной причиной головной боли. Еще не так давно программирование печати в Windows-приложении всеми признавалось высшим пилотажем, доступным только особо одаренным. Однако теперь благодаря таким средствам разработки, как MFC, эта задача вполне по плечу любому, кто хочет ее решить.

Что касается вывода на печать, библиотека MFC располагает такими мощными средствами, что для вывода одностраничного документа вам практически ничего не нужно делать самому. Можете поверить мне на слово, но, если ходите убедиться, давайте сотворим простенькое приложеньице; которое будет поддерживать функции печати и предварительного просмотра распечатки документа. Итак, следуйте за мной!

1. Выберите File⇨New, затем вкладку Projects. Установите имя проекта `Print1` и соответствующие каталоги для файлов проекта. Проверьте, чтобы был выбран вариант `MFC AppWizard (exe)` в левом окне. Щелкните на OK.

2. Пройдитесь по всем диалоговым окнам `AppWizard`, заказывая параметры в соответствии с приведенным ниже списком, каждый раз щелкая на `Next`.

   Этап 1. Выберите Single document.

   Этап 2. Не меняйте настроек, предлагаемых `AppWizard` по умолчанию.

   Этап 3. Не меняйте настроек, предлагаемых `AppWizard` по умолчанию.

   Этап 4. Сбросьте все флажки, кроме Printing (Печать) и Print Preview (Предварительный просмотр распечатки).

   Этап 5. Не меняйте настроек, предлагаемых `AppWizard` по умолчанию.

   Этап 6. Не меняйте настроек, предлагаемых `AppWizard` по умолчанию.

   После этого окно New Project Information (Информация о новом проекте) выведет параметры выполненной настройки. Щелкните на кнопке OK в этом окне, и `ClassWizard` сформирует проект.

3. В окне ClassView разверните класс `CPrint1View` и сделайте двойной щелчок на `OnDraw()`. Отредактируйте текст этой функции — вставьте после комментария, оставленного `AppWizard` (`TODO: add draw code for native data here:` (`СДЕЛАТЬ: сюда вставить текст для данных программы:`)), следующий оператор:

```
pDC->Rectangle(20, 20, 220, 220);
```

Вы уже дважды встречались с функцией `Rectangle()` — в приложении `Recs` в главе 4 и в приложении `Paint1` в главе 5. Включение ее в функцию `OnDraw()` — член класса `CPrint1View` приведет к тому, что программа нарисует прямоугольник. Заданный нами прямоугольник будет иметь размер 200×200 пикселей и его левый верхний угол будет расположен в точке, отстоящей на 20 пикселей от левой границы окна и на 20 пикселей ниже верхней границы рабочего поля окна.



Если вы не читали главу 5 и не совсем хорошо представляете себе, что такое *контекст устройства*, прочитайте ее. Тем, кто не читал главу 4 и имеет некоторые сложности с концепцией *документы/представление*, ее тоже весьма полезно прочитать. В этой главе вам придется перегрузить довольно много виртуальных функций базовых классов и интенсивно поработать с контекстом устройства.

Теперь уж вы точно мне поверите, что создаиное приложение не только может вывести изображение прямоугольника в окне документа на экране, но и в окие предварительного просмотра распечатки, а также и иа приитер. Оттранслируйте и скомпонуйте приложение Print1 (для этого нужно выбрать в меию Build⇨Build или иажать <F7> на клавиатуре). Затем запустите приложение — выберите Build⇨Execute. В результате на экраие появится изображение прямоугольника, как на рис. 6.1. Это и есть выходная информация приложения. Хоть и простенькое, но свое! Теперь выберите File⇨Print Preview. Появится окно предварительиого просмотра распечатки, представлениое на рис. 6.2. Изображение воспроизводит документ, подготовленный приложением в том виде, в котором он будет распечатан. Продолжим испытания и попробуем действительно распечатать наш "документ". Для этого выберите File⇨Print. Эта команда включеиа в меию приложения, поскольку при настройке AppWizard вы установили соответствующую опцию на этапе 4.

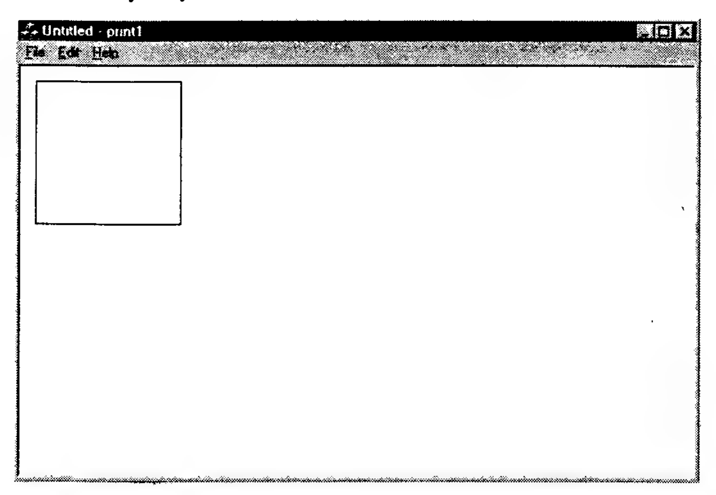


Рис. 6.1. При запуске приложение Print1 выведет на экран изображение прямоугольника

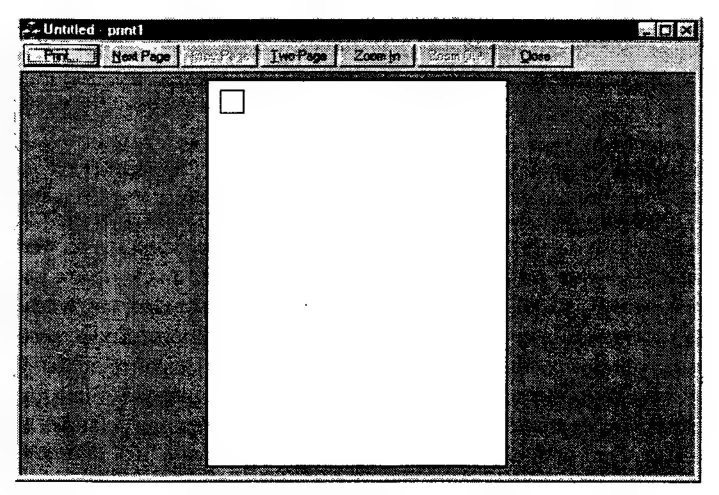


Рис. 6.2. Поскольку при настройке AppWizard была установлена опция Print Preview, приложение Print1 выведет на экран изображение документа в том виде, в котором он будет распечатан

Масштабирование

Рассматривая изображение в окне предварительного просмотра, вы, несомненно, обратили внимание на то, что изображение прямоугольника в нем выглядит как-то мелковато, особенно по сравнению с "родным" окном приложения. Это произошло потому, что пиксель на экране монитора и точка, как ее понимает принтер, имеют разные размеры. Хотя в обоих случаях — и в экранной версии, и в распечатке — прямоугольник имеет стороны равной протяженности, т.е. является квадратом, на листе документа он значительно меньше. Такое соотношение размеров является следствием установки *режима наложения* MM_TEXT, что было сделано в нашем приложении по умолчанию. Если у вас есть необходимость масштабировать изображение таким образом, чтобы оно вписывалось в некоторый заданный размер, нужно выбрать другой режим наложения. В табл. 6.1 перечислены режимы наложения (mapping mode), которые Windows предоставляет в ваше распоряжение.

Таблица 6.1. Режимы наложения

Режим	Единица измерения	Координата	
		X	Y
MM_HIENGLISH	0,001 дюйма	Возрастает слева направо	Возрастает снизу вверх
MM_HIMETRIC	0,01 мм	Возрастает слева направо	Возрастает снизу вверх
MM_ISOTROPIC	Определяется пользователем	Определяется пользователем	Определяется пользователем
MM_LOENGLISH	0,01 дюйма	Возрастает слева направо	Возрастает снизу вверх
MM_LOMETRIC	0,1 мм	Возрастает слева направо	Возрастает снизу вверх
MM_TEXT	Пиксель устройства	Возрастает слева направо	Возрастает сверху вниз
MM_TWIPS	1/1440 дюйма	Возрастает слева направо	Возрастает снизу вверх

Работа с графикой в режиме MM_TEXT порождает определенные проблемы, поскольку принтер и экран имеют разные количества элементов разложения на страницу. С графикой лучше работать в режиме MM_LOENGLISH, который использует в качестве едииц разложения сотые доли дюйма, а не пиксели или точки принтера. Для того чтобы в приложении Paint1 использовать режим наложения MM_LOENGLISH, замените вставленную в функцию OnDraw()строку следующей парой строк:

```
pDC->SetMapMode(MM_LOENGLISH);
pDC->Rectangle(20, -20, 220, -220);
```

Первый оператор включает новый режим наложения в контекст устройства. Второй задает отрисовку прямоугольника с учетом направления осей новой системы координат. Откуда появились отрицательные значения координат? Если вы внимательно посмотрите на табл. 6.1, то заметите, что для режима MM_LOENGLISH ось X направлена слева направо, как и обычно в экранных системах координат, а ось Y направлена снизу вверх, а не наоборот. Более того, по умолчанию экранная система координат располагается в правом нижнем квадранте общепринятой декартовой системы (Cartesian coordinate system), как это показано на рис. 6.3. Если установлен режим наложения MM_LOENGLISH, то в распечатке документа стороны прямоугольника будут иметь длину ровно по 2 дюйма. Это происходит потому, что при значении машинной единицы в 1/100 дюйма задана длина сторон прямоугольника по 200 единиц.

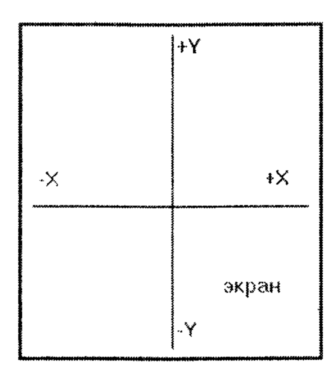


Рис. 6.3. Положение системы координат в режиме наложения MM_LOENGLISH по умолчанию является производным от общепринятой декартовой системы координат

Распечатка многостраничного документа

Когда вы имеете дело с таким простеньким приложением, как Print1, то включение в него функций вывода на печать выполняется практически без вашего участия. Это происходит потому, что документ занимает только одну страницу и, таким образом, никакой разбивки на страницы не требуется. Что бы вы ни включили в такой одностраничный документ (кроме растровых картинок (bitmap image)), MFC все заботы по распечатке берет на себя. Функция OnDraw(), которая является членом класса представления вашего приложения, одинаково формирует изображение документа и для вывода в окно приложения, и для вывода на печать, и для вывода в окно предварительного просмотра распечатки. Однако все значительно усложняется, когда речь заходит о больших документах, которые требуют разбивки на страницы или специальной обработки, как, например, включения верхних и нижних колонтитулов.

Для того чтобы на практике убедиться в наличии проблем, возникающих при распечатке многостраничных документов, модифицируем приложение Print1. Увеличим количество формируемых прямоугольников с тем, чтобы формируемое приложением изображение уже не вмещалось в одну страницу. Теперь у вас появится возможность на практике разобраться с методикой разбивки на страницы. (Такой метод обучения называется *Создать себе трудности, а затем их героически преодолеть.*) Чтобы разнообразить задачу, добавим в класс документа приложения член-переменную, который будет хранить количество рисуемых прямоугольников и позволит пользователю увеличивать или уменьшать их количество, щелкая левой и правой кнопками мыши. Итак, следуйте за нами!

1. В окне ClassView разверните класс CPrint1Doc и щелкните правой кнопкой мыши на нем. Выберите Add Member Variable (Добавить член-переменную) в контекстном меню. В поле Variable Type (Тип переменной) введите int, в поле Variable Declaration (Объявление переменной) введите m_numRecs, а тип доступа задайте Public (Открытая).
2. Дважды щелкните на конструкторе CPrint1Doc и включите в него оператор:

   ```
   m_numRecs = 5;
   ```

   В результате непосредственно после запуска приложения в документе будет сформировано 5 прямоугольников.
3. С помощью ClassWizard организуйте перехват щелчка левой кнопкой мыши (сообщение WM_LBUTTONDOWN) функцией OnLButtonDown() — членом класса представления, как это показано на рис. 6.4.
4. Теперь щелкните на Edit Code (Редактировать текст программы) с тем, чтобы отредактировать текст функции OnLButtonDown(). В листинге 6.1 представлен полный текст обработчика щелчка левой кнопкой мыши. Функция просто увеличивает значение счетчика прямоугольников.

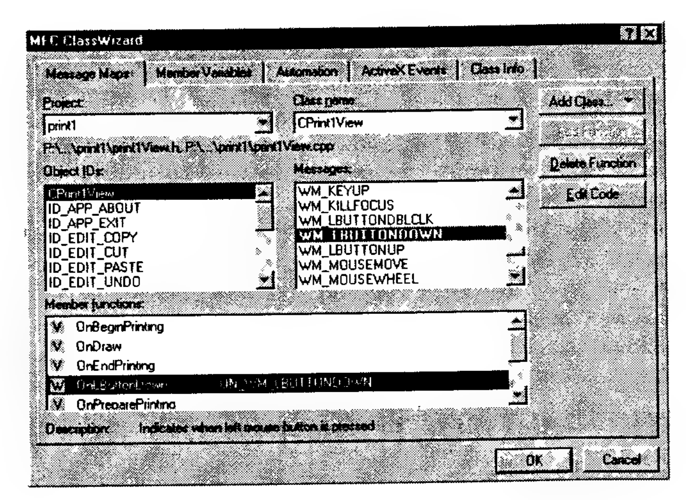


Рис. 6.4. Использование ClassWizard *для включения в приложение функции* OnLButtonDown()

Листинг 6.1. Файл Print1View.cpp **— функция** CPrint1View::OnLButtonDown()

```
void CPrint1View::OnLButtonDown(UINT nFlags, CPoint point)
{
  CPrint1Doc* pDoc = GetDocument();
  ASSERT_VALID(pDoc);

  pDoc->m_numRects++;
  Invalidate();

  CView::OnLButtonDown(nFlags, point);
}
```

5. С помощью ClassWizard добавьте функцию обработки щелчка правой кнопкой мыши OnRButtonDown(), как это показано на рис. 6.5.

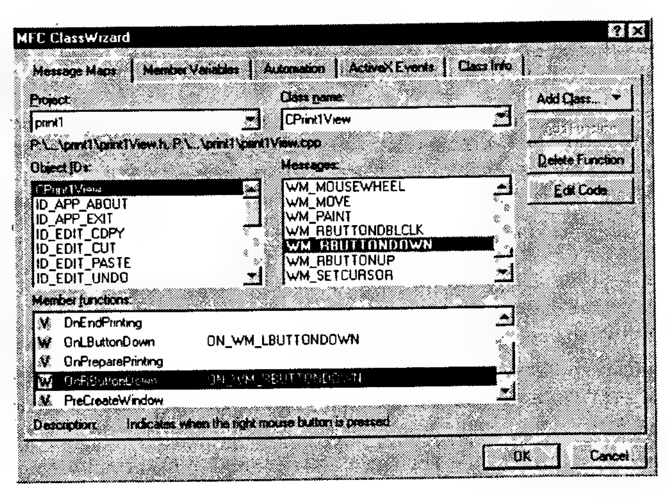


Рис. 6.5. Использование ClassWizard *для включения в приложение функции* OnRButtonDown()

6. Теперь щелкните на **Edit Code** и отредактируйте текст функции `OnRButtonDown()`так, чтобы он повторял текст листинга 6.2. Функция уменьшает счетчик прямоугольников всякий раз, когда пользователь щелкает правой кнопкой мыши.

Листинг 6.2. Файл `Print1View.cpp` — функция `CPrint1View::OnRButtonDown()`

```
void CPrint1View::OnRButtonDown(UINT nFlags, CPoint point)
{
  CPrint1Doc* pDoc = GetDocument();
  ASSERT_VALID(pDoc);

  if (pDoc->m_numRects > 0)
  {
    pDoc->m_numRects--;
    Invalidate();
  }

  CView::OnRButtonDown(nFlags, point);
}
```

7. Модифицируйте функцию `OnDraw()` — член класса представления. Функция должна формировать не один, а несколько — `m_numRecs` — прямоугольников. Новый вариант текста представлен в листинге 6.3. Теперь приложение `Print1` будет формировать прямоугольники один под другим, в результате чего количество страниц документа может возрастать. Кроме того, количество сформированных прямоугольников также выводится на экран.

Листинг 6.3. Файл `Print1View.cpp` — функция `CPrint1View::OnDraw()`

```
void CPrint1View::OnDraw(CDC* pDC)
{
  CPrint1Doc* pDoc = GetDocument();
  ASSERT_VALID(pDoc);

  // TODO: сюда вставить текст для данных программы.
   pDC->SetMapMode(MM_LOENGLISH);

   char s[10];
   wsprintf(s, "%d", pDoc->m_numRects);
   pDC->TextOut(300, -100, s);

   for (int x=0; x<pDoc->m_numRects; ++x)
   {
      pDC->Rectangle(20, -(20+x*200), 200, -(200+x*200));
   }
}
```

После запуска модифицированной версии приложения вы увидите на экране картинку, представленную на рис. 6.6. В изображении присутствуют не только собственно прямоугольники, но и их количество, так что всегда можно определить, сколько страниц документа потребуется для их распечатки (конечно, после несложного пересчета в уме). По команде **File⇨Print Preview** на экран будет выведено окно предварительного просмотра распечатки. Щелкните на кнопке **Two Pages** (Две страницы), и окно предварительного просмотра примет вид, представленный на рис. 6.7. Пять прямоугольников уместились на одной странице, а вторая страница пуста.

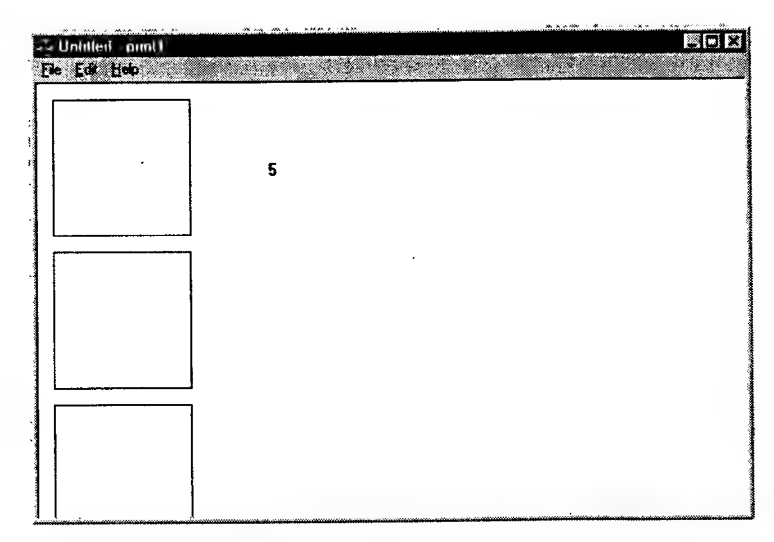


Рис. 6.6. Приложение `Print1` *вывело на экран несколько прямоугольников*

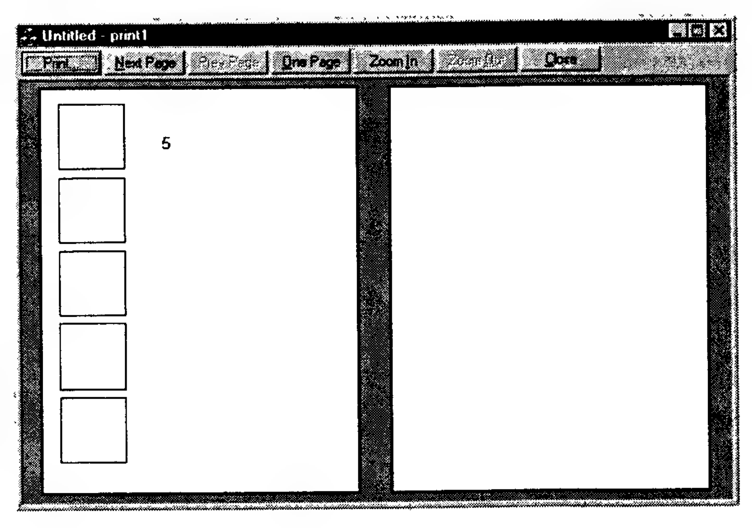


Рис. 6.7. Пять прямоугольников свободно уместились на одной странице, как показано в окне предварительного просмотра

Вернитесь теперь к главному окну приложения и трижды щелкните левой кнопкой мыши на поле окна с тем, чтобы добавить еще три прямоугольника к уже имеющимся пяти. После этого один из них уберите, щелкнув правой кнопкой мыши. Число на экране, соответствующее количеству прямоугольников в документе, должно иметь значение 7. Снова запросите вывод окна предварительного просмотра распечатки. Вы должны увидеть то, что показано на рис. 6.8. Программа еще не знает, как ей поступить с тем изображением, которое не умещается на первой странице. Поэтому часть шестого прямоугольника показана на первой странице, а вторая страница пуста.

Первое, что нужно сделать, чтобы "объяснить" MFC, как правильно распределять содержимое документа между страницами, — вызвать функцию `SetMaxPage()` в методе `OnBeginPrinting()` класса представления приложения. `AppWizard` уже создал для вас заготов-

ку OnBeginPrinting(), но ничего полезного в ней еще нет. Вам придется самостоятельно внести в нее нужные операторы так, чтобы в законченном виде текст функции соответствовал представленному в листинге 6.4.

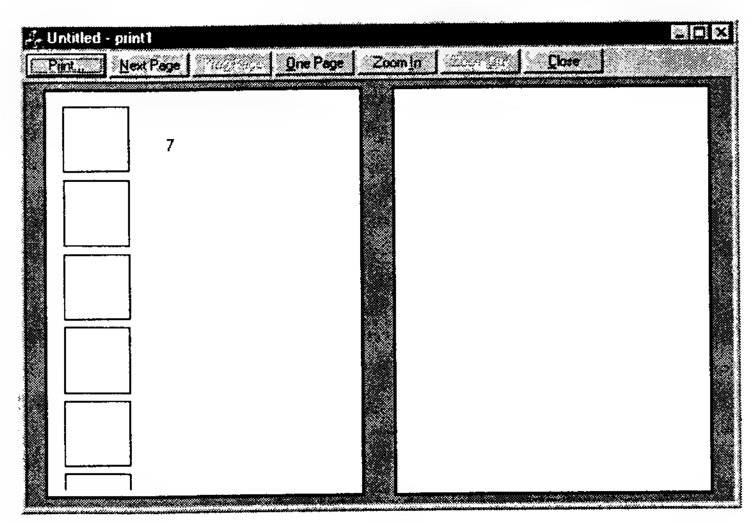


Рис. 6.8. Семь прямоугольников пока не могут быть правильно распределены между несколькими страницами документа

Листинг 6.4. Файл Print1View.cpp — функция CPrint1View::OnBeginPrinting()

```
void CPrint1View::OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo)
{
    CPrint1Doc* pDoc = GetDocument();
   ASSERT_VALID(pDoc);

    int pageHeight = pDC->GetDeviceCaps(VERTRES);
    int logPixelsY = pDC->GetDeviceCaps(LOGPIXELSY);
    int rectHeight = (int)(2.2 * logPixelsY);
    int numPages = pDoc->m_numRects * rectHeight / pageHeight + 1;
    pInfo->SetMaxPage(numPages);
}
```

Метод OnBeginPrinting()принимает два аргумента — указатель контекста устройства (принтера) и указатель на объект класса CPrintInfo. Поскольку в заготовке текста функции нет ссылок на эти аргументы, они “закомментированы”:

```
void CPrint1View::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
```

Это позволяет избежать ненужных предупреждающих сообщений компилятора при трансляции исходной (неотредактированной) версии функции. Однако для того, чтобы установить счетчик страниц, придется обращаться как к объекту класса CDC, так и к объекту класса CPrintInfo. Поэтому первое, что нужно сделать, — это освободить аргументы от ограничителей комментариев.

Теперь вам понадобятся некоторые сведения о контексте устройства, в данном случае — принтера. А именно — нужно знать длину страницы в точках (элементах разложения при печати — dot) и количество точек на дюйм. Длину страницы можно узнать, обратившись к функции GetDeviceCaps() — члену класса CDC. Эта функция возвращает отдельные параметры настройки контекста устройства. Если передать ей в качестве аргумента константу VERTRES, функция вернет количество точек от верхнего до нижнего края страницы. Аргумент

HORZRES заставит ее вернуть размер страницы по горизонтали. Всего существует 29 констант, которые можно использовать как аргументы при обращении к функции GetDeviceCaps(). В частности, NUMFONTS можно использовать для запроса количества поддерживаемых шрифтов, а DRIVERVERSION — для получения номера версии драйвера. Полный список всех констант имеется в оперативной справке Visual C++.

Приложение Print1 применяет в контексте устройства режим иаложения MM_LOENGLISH. Это означает, что при формировании изображения используются линейные единицы, соответствующие 1/100 дюйма. Для того чтобы узнать, сколько прямоугольников может уместиться на одной странице документа, необходимо располагать информацией о длине страницы и высоте прямоугольника, причем оба параметра должны иметь одинаковые единицы измерения (пусть это будут элементы разложения принтера — точки), а затем просто разделить один параметр на другой — и получим искомое. Теперь, надеюсь, вам понятно, почему приложение должно знать о документе все (иначе не рассчитать количество страниц). Мы задали размер прямоугольника по вертикали равным 2 дюйма, а интервал между соседними прямоугольниками — 20/100 дюйма. Таким образом, шаг прямоугольников по вертикали — 2,2 дюйма. Функция GetDeviceCaps() возвратит количество точек принтера на дюйм, если передать ей в качестве аргумента LOGPIXELSY. Умножив это значение на 2,2 получим количество точек на один прямоугольник с учетом интервала.

Наконец-то у нас есть вся информация для того, чтобы определить количество страниц документа, в который входит известное количество прямоугольников. Результат нужно передать функции SetMaxPage() в качестве аргумента и можно, казалось бы, со спокойной совестью считать работу над новой версией CPrint1View::OnBeginPrinting() завершенной.

В который уже раз оттранслируйте и запустите программу. Увеличьте количество прямоугольников до семи. Этот параметр виден на экране в основном окне приложения. Дайте команду File⇨Print Preview и настройте режим просмотра двух страниц распечатки документа. То, что вы увидите, должно весьма походить на то, что показано на рис. 6.9. О Боже! Опять не то! Мы вылезли из одной ямы, чтобы попасть в другую. Хотя приложение и показало прямоугольники на второй странице, но это не совсем то, чего мы добивались. Вторая страница должна начинаться там, где закончилась первая, а в нашем случае вторая страница просто копирует первую, т.е. на ней все начинается сначала. Выходит еще есть над чем поработать...

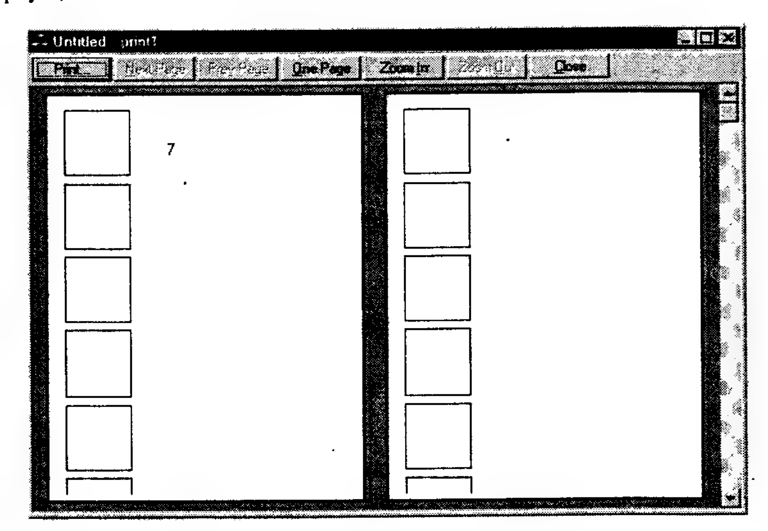


Рис. 6.9. Приложение Print1 все еще не может правильно распределить информацию между страницами документа

Установка начала отсчета

Для того чтобы правильно распечатать вторую и последующие страницы, нужно дать знать MFC, где находится начало страницы (верхний обрез листа). Сейчас MFC выводит изображение на принтер точно так, как вы задали в функции `CPrint1View::OnDraw()`: все прямоугольники (в нашем эксперименте их было семь), начиная с верхнего края страницы. Чтобы объяснить MFC, где находится начало новой страницы, нужно сначала перегрузить функцию `OnPrepareDC()` класса представления.

Вызовите `ClassWizard` и выберите вкладку **Message Maps** (Карты сообщений). Проверьте, чтобы в поле **Class name** (Имя класса) было установлено `CPrint1View`, как это показано на рис. 6.10. Щелкните на `CPrint1View` в списке **Object IDs** (Идентификаторы объектов) и на `OnPrepareDC` в списке **Messages** (Сообщения). После этого щелкните на кнопке **Edit Code** (Редактировать текст программы) и можете приступать к редактированию только что добавленной функции класса `CPrint1View`. Текст этой функции должен соответствовать тексту листинга 6.5.

Листинг 6.5. Файл `Print1View.cpp` — функция `CPrint1View::OnPrepareDC()`

```
void CPrint1View::OnPrepareDC(CDC* pDC, CPrintInfo* pInfo)
{
    if (pDC->IsPrinting())
    {
        int pageHeight = pDC->GetDeviceCaps(VERTRES);
        int originY = pageHeight * (pInfo->m_nCurPage - 1);
        pDC->SetViewportOrg(0, -originY);
    }

    CView::OnPrepareDC(pDC, pInfo);
}
```

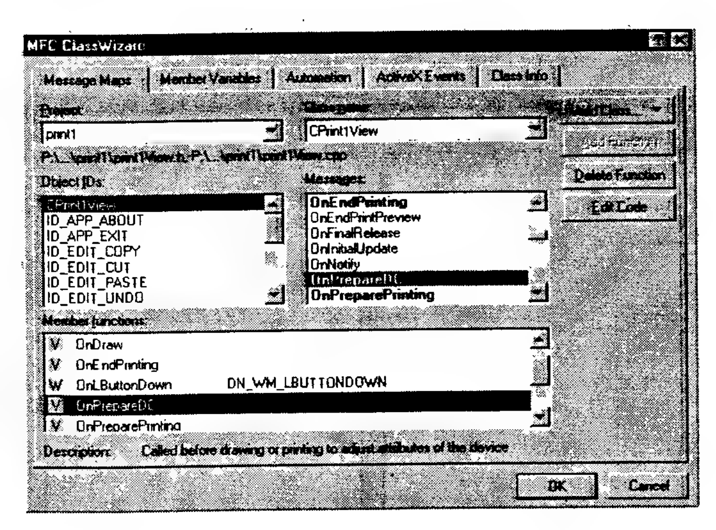


Рис. 6.10. Использование `ClassWizard` для перегрузки функции `OnPrepareDC()`

Системная оболочка MFC вызывает OnPrepareDC() как раз перед выводом данных на экран или их передачей на принтер. (Одно из главных достоинств концепции контекста устройства именно в том и заключается, что одна и та же программа используется и для вывода на экран, и для вывода на печать.) Если приложение собирается выводить данные на экран, вы, скорее всего, не будете ничего менять в процедуре, выполняемой OnPrepareDC() по умолчанию. Так что нужно проверить, не собирается ли приложение выводить данные на печать. Сделать это можно, вызвав метод IsPrinting() класса контекста устройства.

Если приложение готово к распечатке, нужно определить, какая часть документа находится на очередной странице. Для этого придется выяснить длину страницы в элементах разложения принтера, т.е. снова обратиться к GetDaviceCaps().

Далее нужно задать новое начало пользовательской системы координат (системы координат видеопорта — viewport) на поле изображения. Положение пользовательской системы координат позволяет MFC определиться с размещением данных на поле изображения. Для первой страницы начало пользовательской системы координат находится в точке с координатами (0,0). Для второй страницы пользовательская система координат смещается вниз соответственно длине страницы, выраженной в единицах разложения устройства. В общем случае вертикальное смещение пользовательской системы координат есть произведение номера страницы, уменьшенного на единицу, и длины страницы. Не забудьте, что номер страницы хранится в переменной-члене класса CPrintInfo.

Вычислив новое положение начала пользовательской системы координат, нужно передать его в контекст устройства, для чего следует вызвать функцию SetViewportOrg(). На этом завершаются изменения, внесенные в функцию OnPrepareDC().

Чтобы посмотреть, что же из всего этого получилось, оттранслируйте и запустите новую версию приложения Print1. Когда появится основное окно приложения, дважды щелкните на нем левой кнопкой мыши, и в результате количество прямоугольников на экране увеличится до семи. Этот параметр виден на экране. Выполните команду File⇨Print Preview и настройте режим просмотра двух страниц распечатки документа (рис. 6.11). Наконец-то программа правильно разделила документ на страницы! Если сейчас задать вывод на печать, то на бумаге — на первой и второй страницах — будет отпечатано в точности то изображение, которое вы видите в окне предварительного просмотра.

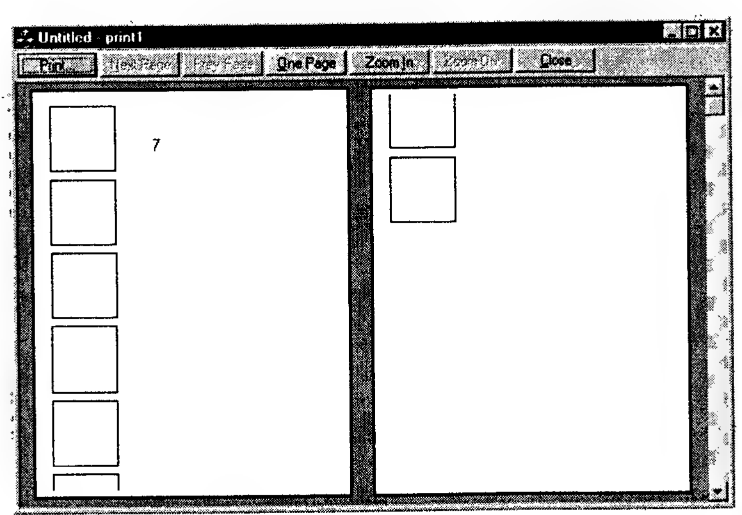


Рис. 6.11. Мы добились своего — приложение Print1, в конце концов, правильно разбило документ на страницы

MFC и вывод на печать

На примере приложения Print1 вы смогли узнать, как в действительности MFC поддерживает функции вывода на печать. По мере того как усложнялось приложение Print1, приходилось перегружать функции-члены класса представления, в том числе OnDraw(), OnBeginPrinting() и OnPrepareDC(). Эти функции играют главную роль в процессе подготовки информации к распечатке. Но ими возможности MFC не исчерпываются. Существует еще несколько функций, которые позволяют расширить функциональные возможности приложения по выводу информации на печать. Все функции, имеющие хоть какое-нибудь отношение к процессу распечатки, перечислены в табл. 6.2. Там же даны и краткие сведения о их назначении.

Таблица 6.2. Функции вывода на печать в классе представления

Функция	Описание
OnBeginPrinting()	Необходимо перегрузить эту функцию в случае, если формируются ресурсы, необходимые для распечатки документа, в частности — шрифты. В этой же функции необходимо установить максимальное значение счетчика количества страниц
OnDraw()	Эта функция несет тройную нагрузку — выводит данные в основное окно приложения, в окно предварительного просмотра распечатки и на принтер в зависимости от контекста устройства, который передается в качестве аргумента вызова
OnEndPrinting()	Необходимо перегрузить эту функцию для того, чтобы освободить ресурсы, сформированные в функции OnBeginPrinting()
OnPrepareDC()	Необходимо перегрузить эту функцию с тем, чтобы модифицировать контекст устройства, которое будет использовано для вывода документа. Здесь, в частности, можно производить разбивку на страницы.
OnPreparePrinting()	Необходимо перегрузить эту функцию с тем, чтобы установить максимальное количество страниц документа. Если этот параметр не будет установлен в функции OnPreparePrinting(), его нужно будет устанавливать в функции OnBeginPrinting()
OnPrint()	Необходимо перегрузить эту функцию с тем, чтобы реализовать дополнительный сервис распечатки, который не обеспечивается функцией OnDraw(), например включение колонтитулов

При выводе документа на печать MFC вызывает перечисленные функции в определенном порядке. Сначала вызывается OnPreparePrinting(), которая просто обращается к DoPreparePrinting(), как показано в листинге 6.6. Последняя организует работу со стандартным диалоговым окном Print и формирует контекст устройства для принтера.

Листинг 6.6. Файл Print1View.cpp — функция CPrint1View::OnPreparePrinting(), подготовленная AppWizard

```
BOOL CPrint1View::OnPreparePrinting(CPrintInfo* pInfo)
{
  // Подготовка по умолчанию.
  return DoPreparePrinting(pInfo);
}
```

Как видно, OnPreparePrinting() получает в качестве аргумента вызова указатель на объект класса CPrintInfo. Этот объект несет в себе информацию о распечатке. Его же можно использовать и для инициализации некоторых параметров, в частности — количества страниц. В табл. 6.3 перечислены основные члены класса CPrintInfo.

Таблица 6.3. Члены класса CPrintInfo

Член	Назначение
SetMaxPage()	Устанавливает номер последней страницы документа
SetMinPage()	Устанавливает номер первой страницы документа
GetFromPage()	Считывает номер начальной страницы документа, которую пользователь определил для вывода на печать
GetMaxPage()	Считывает номер последней страницы документа, который можно будет изменить в функции OnBeginPrinting()
GetMinPage()	Считывает номер начальной страницы документа, который можно будет изменить в функции OnBeginPrinting()
GetToPage()	Считывает номер последней страницы документа, которую пользователь определил для вывода на печать
m_bContinuePrintin g	Управляет процессом печати. Если установить значение этого флага FALSE, распечатка прекратится
m_bDirect	Индикатор режима непосредственной печати документа
m_bPreview	Индикатор режима предварительного просмотра распечатки документа
m_nCurPage	Хранит номер текущей страницы, выводимой на принтер
m_nNumPreviewPages	Хранит количество страниц (1 или 2), которые будут выведены в окно предварительного просмотра распечатки
m_pPD	Хранит указатель на объект класса CPrintDialog
m_rectDraw	Хранит параметры прямоугольника, определяющего рабочую область текущей страницы
m_strPageDesc	Хранит строку форматирования номера страницы

Значения многих членов объекта класса CPrintInfo устанавливаются непосредственно в диалоговом окне Print. После этого программа может либо считывать установленные значения переменных, либо самостоятельно их изменять. В стандартной последовательности вызывается, как минимум, функция SetMaxPage(), которая устанавливает максимальное количество страниц документа еще до обращения к DoPreparePrinting(). В результате максимальное количество страниц документа высвечивается уже в диалоговом окне Print. Если же не удается подсчитать количество страниц документа, поскольку неизвестна настройка конкретного принтера (длина страницы), придется подождать, пока будет сформирован соответствующим образом контекст устройства.

После DoPreparePrinting() MFC вызывает OnBeginPrinting(). В ходе выполнения этой функции можно не только дублировать (или корректировать) установку максимального количества страниц документа, но и формировать необходимые ресурсы, в частности — шрифт. Аргументами OnBeginPrinting() являются указатели контекста устройства (принтера) и указатель на ассоциированный с ним объект класса CPrintInfo.

Далее MFC вызывает OnPrepareDC() для первой страницы документа. Это начало цикла печати, который выполняется для каждой страницы документа. Именно в функции OnPrepareDC() определяется, какая часть всего документа находится на данной странице. Как было сказано выше, эта задача решается путем установки новых значений начала отсчета пользовательской системы координат.

После OnPrepareDC() наступает очередь OnPrint(), которая и выводит на печать текущую страницу. Как правило, OnPrint() вызывает OnDraw(), передавая ей в качестве аргумента указатель контекста устройства — принтера. Это автоматически направляет данные не на экран, а на принтер. Но можно и перегрузить функцию OnPrint(), чтобы реализовать собственный

алгоритм управления выводом на печать. В ходе выполнения этой функции можно организовать вывод колонтитулов (верхнего и нижнего), а уже затем вызвать одноименный метод базового класса (который, в свою очередь, вызовет OnDraw()) для распечатки основного содержимого документа. Текст такого варианта функции OnPrint() приведен в листинге 6.7. (Нижний колонтитул всегда будет выводиться *под* основным содержимым страницы документа, даже если функция PrintFooter() вызывается *перед* OnPrint(). Об этом можно не беспокоиться.) Предотвратить стирание колонтитулов основным содержимым документа в ходе выполнения метода базового класса можно, ограничив рабочую зону страницы. Для этого следует установить соответствующее значение переменной m_rectDraw — члена объекта класса CPrintInfo.

Листинг 6.7. Вариант функции OnPrint() с распечаткой колонтитулов

```
void CPrint1View::OnPrint(CDC* pDC, CPrintInfo* pInfo)
{
   // TODO: сюда вставьте свой фрагмент текста
   //  и/или вызов базового класса.
   // Вызов локальных функций для печати колонтитулов.
   PrintHeader();
   PrintFooter();

   CView::OnPrint(pDC, pInfo);
}
```

Можно вообще убрать OnDraw() из цикла печати и организовать собственную процедуру вывода информации, которая будет работать *только при выводе на печать* (листинг 6.8).

Листинг 6.8. Вариант функции OnPrint() без обращения к OnDraw()

```
void CPrint1View::OnPrint(CDC* pDC, CPrintInfo* pInfo)
{
     // TODO: сюда вставьте свой фрагмент текста
     //  и/или вызов базового класса.
     // Вызов локальных функций для печати колонтитулов.
   PrintHeader();
   PrintFooter();

     // Вызов локальной функции распечатки документа.
   PrintDocument();
}
```

Для распечатки каждой страницы MFC будет продолжать поочередно вызывать OnPrepareDC() и OnPrint(). После вывода последней страницы MFC вызовет OnEndPrinting(). В этой функции можно освободить все ресурсы, сформированные в OnBeginPrinting(). Полностью алгоритм вывода на печать, который реализуется в MFC, представлен на рис. 6.12.

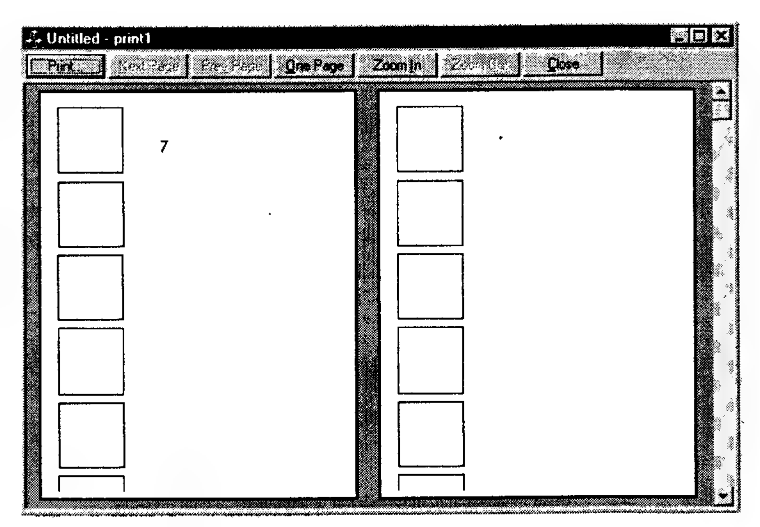


Рис. 6.12. В процессе вывода данных на печать MFC вызывает различные функции-члены класса представления

ГЛАВА 7

Сохранение-восстановление объектов и работа с файлами

В этой главе...

Концепция сохранения-восстановления объектов

Структура приложения File Demo

Создание класса, обеспечивающего сохранение-восстановление объектов

Непосредственное чтение и запись файлов

Создание объекта класса CArchive

Системный реестр Registry

Концепция сохранения-восстановления объектов

Ни для кого не секрет, что одна из основных задач программы — сохранять данные пользователя после их изменения по той или иной причине. Без этого все усилия, которые пользователь затратил на редактирование данных, пропадут, как только приложение завершит работу. С таким инструментом кашу не сваришь! В большинстве случаев, когда приложение создается с использованием AppWizard, Visual C++ без вашего участия включает в него программы, которые необходимы для сохранения и восстановления данных. Однако иногда, точнее — когда вы создаете собственные типы объектов, придется кое-что сделать самостоятельно, чтобы обеспечить надежное сохранение-восстановление пользовательских данных.

При создании приложения вам приходится иметь дело с достаточно большим разнообразием типов объектов. Одни типы объектов, хранящих данные, довольно просты, например тип int или char. Другие являются экземплярами классов — строками (экземплярами класса CString) или даже объектами классов, созданными специально для данного приложения. При использовании таких объектов в приложении, которое должно формировать, сохранять и восстанавливать документы, разработчику волей-неволей необходимо изобретать средства сохранения и восстановления этих объектов с тем, чтобы можно было их восстановить.

Свойство объекта сохраняться и восстанавливаться называется *живучестью* (persistence). Практически все классы MFC наделены этим свойством, поскольку они прямо или косвенно происходят от базового класса CObject. Последний уже обладает базовыми функциями сохранения-восстановления объекта. В последующих разделах этой главы вы узнаете, как MFC обеспечивает живучесть объекта класса документа.

Структура приложения File Demo

При создании программы с помощью AppWizard вы получите приложение, которое использует классы документа и представления для формирования, редактирования и отображения данных. Как было показано в главе 4, объект класса документа, производного от CDocument, отвечает за хранение данных в течение всего сеанса работы приложения, а также за сохранение и загрузку данных, так что документ сохраняет свое состояние после завершения одного сеанса работы с ним и восстанавливает в начале следующего сеанса.

В процессе изучения этой главы вы создадите приложение File Demo, которое демонстрирует основные методы сохранения и восстановления данных из объектов классов, производных от CDocument. Документ этого приложения представляет собой единственную текстовую строку с кратким, но многозначительным сообщением, которая выводится на экран классом представления.

Основную роль в работе приложения File Demo играют три команды меню. При первом запуске текст сообщения автоматически устанавливается как Default Message (Сообщение по умолчанию). Его можно изменить, выполнив команду меню Edit⇨Change Message. Сохранить документ можно с помощью команды File⇨Save, а вновь загрузить — с помощью File⇨Open.

Классы документа

Способность объектов сохраняться после изменения в одном сеансе и восстанавливать свое состояние в следующем, используя файл в качестве промежуточной среды хранения, есть ничто иное, как *живучесть* объектов с точки зрения пользователя. Хотя некоторое представление о классах документов вы уже получили в главе 4, сейчас мы вернемся к основным

концепциям их структуры, фокусируя внимание на возможности реализации этих концепций в классах, создаваемых вами самостоятельно.

Работая с приложениями, созданными с помощью AppWizard, вы должны выполнить некоторую последовательность операций, чтобы обеспечить возможность сохранения и восстановления документа. Эти операции в том виде, в котором они применяются для SDI-приложения (однодокументного приложения), будут рассмотрены в настоящем разделе.

1. Создание членов-переменных класса документа, которые будут хранить специфические для данного вида документа данные.
2. Инициализация этих членов-переменных в методе OnNewDocument()класса документа.
3. Организация отображения текущего состояния документа в методе OnDraw()класса представления.
4. Включение в класс представления методов, обеспечивающих редактирование документа.
5. Модификация метода Serialize() класса документа — включение в него операторов, обеспечивающих сохранение и загрузку данных, которые и представляют собой содержание документа.

Что касается MDI-приложений, то, помимо перечисленных операций, придется сделать кое-что дополнительно, поскольку нужно обеспечить правильный выбор сохраняемого и корректную загрузку восстанавливаемого документов с учетом того, что приложение такого типа работает в течение одного сеанса с множеством документов одновременно. К счастью, большую часть этих функций MFC реализует без участия программиста.

Создание приложения File Demo

Для создания приложения File Demo запустите AppWizard и настройте его на создание SDI-приложения. При этом можно оставить все настройки в том виде, в котором AppWizard их предлагает по умолчанию. Поэтому можно иа этапе 1 сразу после выбора типа приложения щелкнуть на кнопке Finish. Только обратите внимание, чтобы был установлен флажок поддержки архитектуры *документ/представление*.

Щелкните дважды на CFileDemoDoc в окне ClassView и отредактируйте файл заголовка этого класса документа. Добавьте в секцию атрибутов определение переменной m_message типа CString, чтобы этот фрагмент определения класса выглядел следующим образом:

```
// Атрибуты.
public:
  CString m_message;
```

В данном случае документ содержит единственный объект класса CString — строку текста. В реальных приложениях данные будут значительно сложнее. Однако и этой единственной строки текста хватит, чтобы продемонстрировать особенности технологии обеспечения сохранности документа. Очень часто программисты используют в классе документа открытые члены-переменные вместо закрытых членов, для каждого из которых организуется открытая функция доступа. Это несколько облегчает разработку класса представления, методы которого должны обращаться к членам класса документа. Но в дальнейшем при сопровождении программы и, в частности, ее модификации такой подход несколько усложнит жизнь. Более подробно доводы в пользу каждого из альтернативных подходов обсуждаются в приложении А.

Класс документа также должен обеспечить инициализацию данных при открытии нового документа, что возлагается на метод OnNewDocument()этого класса. Вызовите при помощи окна ClassView текст этой функции в окно редактора кода и отредактируйте его. Добавьте оператор инициализации строковой переменной, как в листинге 7.1.

Листинг 7.1. Инициализация данных документа

```
BOOL CFileDemoDoc::OnNewDocument()
{
  if (!CDocument::OnNewDocument())
    return FALSE;

  m_message = "Default Message";

  return TRUE;
}
```

После того как переменная m_message — член класса документа — инициализирована, приложение должно вывести содержимое документа в свое окно. Здесь за дело берется метод OnDraw() класса представления. Вывести текст этой функции в окно редактора кода можно уже известным вам способом при помощи окна ClassView. После редактирования функция должна иметь вид, представленный в листинге 7.2.

Листинг 7.2. Вывод на экран данных документа

```
void CFileDemoView::OnDraw(CDC* pDC)
{
  CFileDemoDoc* pDoc = GetDocument();
  ASSERT_VALID(pDoc);

  pDC->TextOut(20, 20, pDoc->m_message);
}
```

Методика вывода на экран с использованием контекста устройства обсуждалась выше, в главе 5.

Теперь оттранслируйте приложение File Demo и запустите его на выполнение. Вы увидите, что на экране появится сообщение Default Message.

До тех пор, пока содержимое документа удовлетворяет пользователя, программе ничего не надо делать. Но, естественно, приложение, которое не позволяет пользователю редактировать документ, представляет ценность разве что для демонстрации. Для того чтобы обеспечить такую возможность, нужно включить в меню Edit приложения пункт Change Message. По этой команде должны запускаться средства, позволяющие пользователю изменить текст документа — выводимого сообщения. Приложение ShowString, которое будет подробно рассмотрено в главе 8, продемонстрирует, как можно сформировать диалоговое окно (именно такое, в котором нуждается наше приложение File Demo).

Щелкните на вкладке Resource в левой части экрана и вызовите окно ResourceView, разверните компонент Menus и дважды щелкните на IDR_MAINFRAME. Теперь можно приступить к редактированию меню. Щелкните на пункте Edit и разверните его. Щелкните на пустом поле внизу списка пунктов этого меню и введите Change &Message. В результате в меню будет добавлен новый пункт.

Теперь вызовите мастер ClassWizard (команда меню View⇨ClassWizard). Он понадобится вам для установления связи между новой командой и текстом программы. В левом окне ClassWizard должен быть подсвечен пункт ID_EDIT_CHANGEMESSAGE. Если это не так, щелкните на этом пункте. В раскрывающемся списке справа вверху выберите CFileDemoView. Щелкните на COMMAND в окне справа внизу, а затем — на кнопке Add Function. Предлагаемое мастером имя функции OnEditChangemessage() нам вполне подходит, так что щелкните на кнопке OK в появившемся диалоговом окне. Теперь щелкните на кнопке Edit Code — заготовка новой функции будет выведена в окне редактора кода. Ее нужно отредактировать соответственно листингу 7.3.

Листинг 7.3. Редактирование содержимого документа

```
void CFileDemoView::OnEditChangemessage()
{
   CTime now = CTime::GetCurrentTime();
   CStrind changetime = now.Format("Changed at %B %d %H:%M:%S");
   GetDocument()->m_message = changetime;
   GetDocument()->SetModifiedFlag();
   Invalidate();
}
```

Эта функция, которая вызывается после выполнения команды Edit⇨Change Message, формирует строку соответственно текущей дате и времени и присваивает ее переменной-члену текущего объекта класса документа. (Класс CTime и его метод Format() рассматриваются более подробно в приложении E.) Вызов метода SetModifiedFlag() класса документа сообщит объекту, что его содержимое изменено. Если такое изменение зафиксировано, приложение будет предупреждать пользователя о наличии несохраненных изменений в текущем документе при попытке его закрыть. И последняя операция — запуск механизма обновления представления документа на экране, который производится функцией Invalidate(), о чем уже шла речь в главе 4.



Если m_message является закрытой переменной-членом класса документа, понадобится разработать открытый метод SetMessage(), который будет самостоятельно вызывать SetModifiedFlag(). Таким образом, вы будете навсегда избавлены от необходимости напоминать программистам об обязательном обращении к этой функции при модификации объекта класса документа. В этом и состоит преимущество скрупулезного следования принципам объектно-ориентированного программирования.

Метод Serialize()класса документа должен позаботиться о сохранении-восстановлении данных документа. В листинге 7.4 представлен текст заготовки функции Serialize(), сформированный AppWizard.

Листинг 7.4. Заготовка функции Serialize()класса документа

```
void CFileDemoDoc::Serialize(CArchive& ar)
{
   if (ar.IsStoring())
   {
      // TODO: сюда вставьте операторы сохранения данных.

   }
   else
   {
      // TODO: сюда вставьте операторы загрузки данных.

   }
}
```

Поскольку в классе CString (объектом которого является переменная m_message) определены терминальные операторы >> и << для передачи данных *в* архив и *из* него, это значительно упрощает сохранение и восстановление данных в объекте класса документа. Добавьте следующий оператор в том месте, где в заготовке стоит инструктирующий комментарий:

```
ar << m_message;
```

Аналогично в том месте текста программы, где должны стоять операторы загрузки, вставьте

```
ar >> m_message;
```

Терминальный оператор << пересылает CString m_message в архив, а терминальный оператор >> заносит данные в m_message из архива. До тех пор, пока данные документа сохраняются в члене — простой переменной (наподобие int или char) или в объекте такого класса, как CString, для которых определены соответствующие терминальные операторы, сохранение и восстановление документа выполняются очень просто. Указанные терминальные операторы определены для следующих простых типов данных.

- BYTE
- WORD
- int
- LONG
- DWORD
- float
- double

Оттранслируйте приложение File Demo и запустите его. Выберите в меню приложения Edit⇨Change Message и убедитесь, что на экране появилась новая строка сообщения, как показано на рис. 7.1. Теперь выберите File⇨Save и введите имя файла. Опять измените текст сообщения с помощью команды Edit⇨Change Message. Выберите File⇨New — на экране появится предупреждающее сообщение о наличии в документе несохраненных изменений. Вам будет предложено сохранить их на диске прежде, чем открывать новый документ. Теперь выберите File⇨Open и введите имя ранее созданного файла документа (его можно найти и в списке в самом низу меню File). После этого на экране появится ранее сохраненный текст сообщения. Таким образом, вы можете убедиться, что приложение File Demo сохранило документ по вашей команде, а затем восстановило его в прежнем виде.

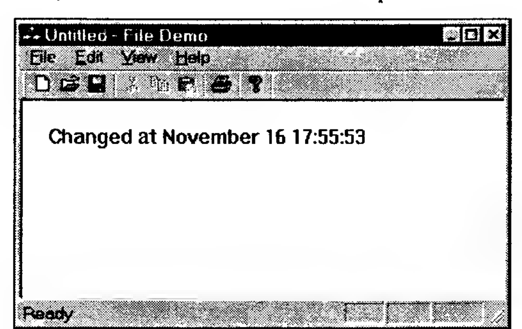


Рис. 7.1. Приложение File Demo изменило текст сообщения по команде пользователя

На заметку Если в документ внесены изменения, он сохранен в файле, в него внесены новые изменения и вызвана команда открыть файл с тем же именем, приложение File Demo не будет выводить запрос Revert to saved document? (Вернуться к сохраненному документу?), как это делают другие программы. Вместо этого перед вами на экране будет самая последняя версия документа. Такой механизм встроен теперь в MFC. Если имя открываемого файла соответствует имени текущего, вы не сможете вернуться к прежней версии документа.

Создание класса, обеспечивающего сохранение-восстановление объектов

Что произойдет, если вы создадите свой собственный класс, объект которого войдет в документ? Как сделать этот объект живучим (в оговоренном выше смысле)? Ответы на эти вопросы вы найдете в настоящем разделе.

Предположим, что вы хотите усовершенствовать приложение File Demo таким образом, чтобы для хранения данных документа использовался объект изобретенного вами класса CMessages. Пусть теперь член класса документа, в котором содержатся данные, называется m_messages и он является экземпляром класса CMessages. Этот класс содержит три объекта класса CString, каждый из которых должен сохраняться и восстанавливаться при сохранении и загрузке документа. Первый способ обеспечить такую возможность — индивидуально сохранять и восстанавливать каждую отдельную строку. Текст соответствующей программы представлен в листинге 7.5.

Листинг 7.5. Возможный вариант сохранения и восстановления строк нового класса

```
void CFileDemoDoc::Serialize(CArchive& ar)
{
  if (ar.IsStoring())
  {
    ar << m_messages.m_message1;
    ar << m_messages.m_message2;
    ar << m_messages.m_message3;
  }
  else
  {

    ar >> m_messages.m_message1;
    ar >> m_messages.m_message2;
    ar >> m_messages.m_message3;
  }
}
```

Такую программу можно написать в том случае, если, во-первых, все три члена класса CMessages являются открытыми (объявлены как public) и, во-вторых, известна реализация самого класса. Если в дальнейшем класс CMessages будет некоторым образом изменен, придется переписать и эту функцию. Если же вы до конца верны моральному кодексу строителя объектно-ориентированных программ, то непременно согласитесь, что функции сохранения и восстановления данных должны быть *инкапсулированы* в самом классе CMessages. Конечно, это потребует определенной предварительной подготовки. Основные этапы создания класса, который может самостоятельно организовать сохранение-восстановление (в документации на MFC применяется термин *serialize — сериализация*) собственных членов-переменных, перечислены ниже.

1. Объявите класс как производный от CObject.
2. В объявление класса включите макрос DECLARE_SERIAL.
3. В реализацию класса включите макрос IMPLEMENT_SERIAL.
4. Перегрузите метод Serialize(), унаследованный от базового класса.
5. Объявите для нового класса среди прочих "пустой" конструктор по умолчанию.

Далее в этом разделе мы проанализируем приложение, в котором именно по этой схеме создаются объекты, обладающие свойством живучести.

Приложение MultiString

В приложении MultiString продемонстрирована та последовательность действий по созданию нового класса, о которой шла речь выше. Это приложение будет иметь команду меню Edit⇨Change Messages, которая изменяет все три строки. Как и приложение File Demo, новая программа сможет сохранять и восстанавливать документы, для чего будут использованы пункты Save и Open меню File.

Для создания приложения MultiString запустите AppWizard и настройте его на тип приложения SDI, точно так, как это делалось при создании File Demo. Добавьте в секцию атрибутов класса документа определение переменной, чтобы этот фрагмент файла MultiStringDoc.h выглядел следующим образом:

```
// Атрибуты.
public:
  CMessages m_messages;
```

Следующий шаг — создание класса CMessages.

Класс CMessages

Прежде чем перейти к классу документа, рассмотрим подробно класс CMessages, объект которого — m_messages — и составляет содержимое документа. Ознакомившись с этим классом, вы поймете, как на практике реализуется перечисленная выше последовательность создания класса, наделенного живучестью.

Для создания этого класса нужно выбрать в меню среды разработки Insert⇨New Class. В поле со списком Class type диалогового окна New Class выберите Generic Class (порождающий класс). В поле Name введите имя нового класса CMessages. Ниже введите имя базового класса CObject, а в столбце As оставьте предлагаемую по умолчанию установку public (рис. 7.2).

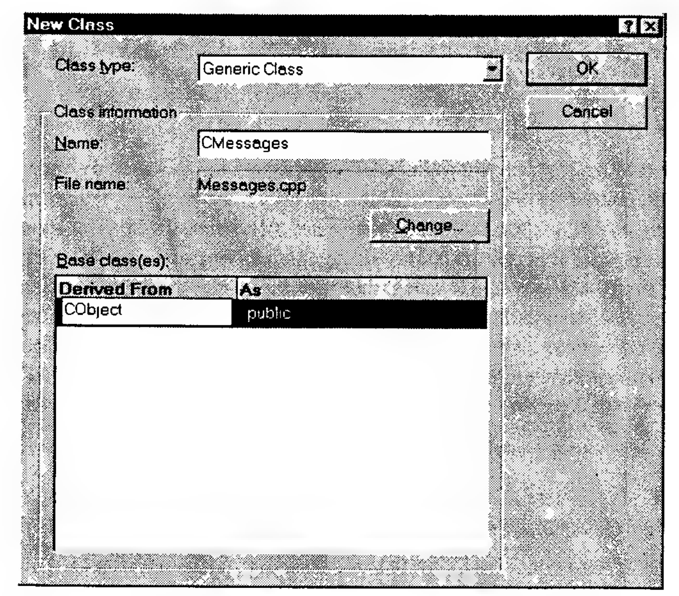


Рис. 7.2. Создание нового класса для сообщений

В результате будут созданы два файла — файл заголовка Messages.h и файл реализации Messages.cpp. В каждый из них будут введены заготовки программного кода. Возможно, будет выведено предупреждающее сообщение о том, что среда разработки не может найти файл заголовка для класса CObject. Если такое случится, можете смело щелкать на OK, поскольку все системные файлы заголовков MFC всегда доступны компилятору без включения директив.

Переключитесь в окне редактора кода на файл MultiStringDoc.h и добавьте в него перед определение класса директиву:

```
#include "Messages.h"
```

В результате при трансляции класса документа компилятору будет доступно определение класса CMessages. Чтобы удостовериться в том, что ничего в определениях не забыто, можно запустить компиляцию проекта. Теперь вернитесь к тексту в файле Messages.h и добавьте следующие строки:

```
DECLARE_SERIAL(CMessages)

protected:
   CString m_message1;
   CString m_message2;
   CString m_message3;

public:
   void SetMessage(UINT msgNum, CString msg);
   CString GetMessage(UINT msgNum);
   void Serialize(CArchive& ar);
```

Макрос DECLARE_SERIAL() обеспечивает выполнение ряда дополнительных функций — MFC использует этот макрос для того, чтобы включить в определение класса объявления дополнительных членов (переменных и функций), которые необходимы для реализации живучести объектов класса.

Далее объявлены члены-переменные класса: три объекта класса CString. Обратите внимание, что они объявлены как защищенные — protected. Следующими идут открытые (public) функции-члены. SetMessage() имеет два аргумента — индекс строки, в которую заносятся данные, и собственно записываемая строка символов. Эта функция позволяет изменить состояние объекта класса. GetMessage() — это сопряженная с предыдущей функция, которая позволяет программе прочесть содержимое любой строки объекта. Ее единственный аргумент — индекс считываемой строки. И наконец, в нашем классе перегружается метод Serialize(), в котором и заключено все таинство сохранения и восстановления данных. Эта функция — сердце живучего объекта. Реализация ее уникальна для каждого класса. В листинге 7.6 представлен текст файла реализации методов класса, в котором вы найдете все его функции-члены.

Листинг 7.6. Файл MESSAGES.CPP — реализация методов класса CMessages

```
void CMessages::SetMessage(UINT msgNum, CString msg)
{
  switch (msgNum)
  {
  case 1:
    m_message1 = msg;
    break;

  case 2:
    m_message2 = msg;
    break;
```

```
  case 3:
     m_message3 = msg;
    break;
  }
}

CString CMessages::GetMessage(UINT msgNum)
{
   switch (msgNum)
  {
  case 1:
      return m_message1;
  case 2:
      return m_message2;
   case 3:
      return m_message3;
   default:
      return "";
  }
}

void CMessages::Serialize(CArchive& ar)
{
   CObject::Serialize(ar);

   if (ar.IsStoring())
   {
      ar << m_message1 << m_message2 << m_message3;
   }
   else
   {
      ar >> m_message1 >> m_message2 >> m_message3;
   }
}
```

В функциях SetMessage() и GetMessage() нет никаких фокусов. Они простым "казачьим" способом реализуют то, что от них требуется. А вот по поводу функции Serialize() могут появиться совершенно законные вопросы. Сначала обратите внимание на то, что первый оператор в теле функции — это вызов одноименного метода базового класса CObject. Это стандартный прием, который используется во многих функциях, перегружающих одноименные функции базового класса. В данном случае — это просто дань дисциплине, поскольку метод CObject::Serialize() ничего не выполняет (он пуст). Но, тем не менее, воспитайте в себе привычку всегда вызывать одноименный метод класса-родителя, поскольку не всегда класс происходит *прямо* от CObject (является *прямым наследником* CObject) и Бог его знает, что делает Serialize()в каком-нибудь другом классе, ребенком которого вы решите объявить свой класс.

После вызова родительской версии Serialize() сохраняет или восстанавливает данные так же, как мы это делали в классе документа. Поскольку члены данных, которые нужно сохранять-восстанавливать, есть объекты класса CString, можно использовать терминальные операторы >> и << для передачи данных *на* диск и *с* диска.

В самом начале файла Messages.cpp после директив include вставьте строку

```
IMPLEMENT_SERIAL(CMessages, CObject, 0)
```

Макрос IMPLEMENT_SERIAL() сопряжен с DECLARE_SERIAL(). Он обеспечивает реализацию функций, которые и обеспечивают живучесть объектов класса. Три аргумента макроса (имя класса, имя класса-родителя и *номер схемы* (schema number)) — это нечто вроде номера версии. В большинстве случаев в качестве номера схемы можно задавать 0 или 1.

Использование класса CMessages в программе

Теперь, когда класс CMessages создан, к нему смогут обращаться методы классов документа и представления приложения MultiString. Разверните в окне ClassView класс CMultiStringDoc и, дважды щелкнув на имени метода OnNewDocument(), вызовите его текст в окне редактора кода. Вместо строк комментария TODO вставьте следующие строки:

```
    m_messages.SetMessage(1, "Default Message 1");
    m_messages.SetMessage(2, "Default Message 2");
    m_messages.SetMessage(3, "Default Message 3");
}
```

Поскольку методы класса документа не могут напрямую обращаться к защищенным членам класса CMessages, каждый из них инициализируется функцией SetMessage() этого класса.

Теперь разверните в окне ClassView класс CMultiStringView и, дважды щелкнув на имени метода OnDraw(), вызовите его текст в окне редактора кода. После редактирования текст этого метода должен выглядеть следующим образом:

```
void CMultiStringView::OnDraw(CDC* pDC)
{
  CMultiStringDoc* pDoc = GetDocument();
  ASSERT_VALID(pDoc);

  pDC->TextOut(20, 20, pDoc->m_messages.GetMessage(1));
  pDC->TextOut(20, 40, pDoc->m_messages.GetMessage(2));
  pDC->TextOut(20, 60, pDoc->m_messages.GetMessage(3));
}
```

Так же, как и при работе с File Demo, добавьте пункт **Change Messages** в меню **Edit**. Подключите его к методу OnEditChangemessages() класса представления. Этот метод будет изменять содержимое документа, обращаясь к методам доступа и к переменным класса CMessages, как показано в листинге 7.7.

Листинг 7.7. Редактирование содержимого документа

```
void CMultiStringView::OnEditChangemessages()
{
   CMultiStringDoc* pDoc = GetDocument();
   CTime now = CTime::GetCurrentTime();
   CStrind changetime = now.Format("Changed at %B %d %H:%M:%S");

   pDoc->m_messages.SetMessage(1, Cstring("String 1 ") + changetime);
   pDoc->m_messages.SetMessage(2, Cstring("String 2 ") + changetime);
   pDoc->m_messages.SetMessage(3, Cstring("String 3 ") + changetime);
   pDoc->SetModifiedFlag();
   Invalidate();
}
```

Существенную работу предстоит выполнить с функцией Serialize() — членом класса документа приложения, в котором данные из объекта m_messages должны записываться-считываться с диска. Это выполняется на удивление просто — вызывается одноименная функция-член класса, которому принадлежит объект, содержащий данные документа, как и показано в листинге 7.8.

Листинг 7.8. Сохранения и восстановления объекта данных

```
void CMultiStringDoc::Serialize(CArchive& ar)
{
  m_messages.Serialize(ar);

  if (ar.IsStoring())
  {
    // TODO: сюда вставьте операторы сохранения данных.
  }
  else
  {
    // TODO: сюда вставьте операторы загрузки данных.
  }
}
```

Как видно, после вызова метода `Serialize()` объекта `m_messages` функции `Serialize()` — члену класса документа приложения больше и делать нечего. Обратите внимание, что при вызове `m_messages.Serialize()` в качестве единственного аргумента передается ссылка на объект класса `CArchive`.

Непосредственное чтение и запись файлов

Хотя использование встроенной в MFC технологии сохранения-восстановления данных и обеспечивает основные требования приложения при работе с файлами, иногда необходимо реализовать нестандартные процедуры управления файловой системой, не укладывающиеся в рамки этой технологии. Например, может возникнуть необходимость вывести информацию в файл, из которого не нужно снова считывать данные в объекты программы, или тот способ записи-считывания, который реализуется в функции `Serialize()`, вас не устраивает, поскольку там возможен только последовательный доступ к файлу (ввод-вывод в поток). В этих случаях можно использовать знакомые всем, кто когда-либо программировал в среде DOS, методики программирования файловых операций — непосредственное создание файла, чтение и запись информации в файл. Но даже если вы и решитесь снизойти до такого уровня программирования задач управления файлами, средства MFC все равно облегчат вам жизнь. Для непосредственного управления процессами ввода-вывода в файл в распоряжение программиста MFC предоставляет класс `CFile` и производные от него.

Класс `CFile`

Включенный в состав MFC класс `CFile` инкапсулирует все функции, связанные с обработкой файлов любого типа. Собираетесь ли вы использовать обычный последовательный способ записи-чтения данных или организовать файл с произвольным доступом к данным, в любом случае можно использовать методы класса `CFile`. При этом последовательность операций очень напоминает прежние С-программы, за исключением того, что класс скрывает некоторые их детали. В результате несколько снижается количество операторов, но принципиально это прежний хорошо знакомый подход. Теперь, в частности, создать файл для чтения можно, используя единственный оператор. В табл. 7.1 перечислены методы класса `CFile` и дано их краткое назначение.

Таблица 7.1. Методы класса CFile

Функция	Назначение
Конструктор	Создает экземпляр класса CFile. Если передать аргумент, имя файла открывает заданный файл
Деструктор	Уничтожает экземпляр класса CFile. Если соответствующий файл открыт, закрывает его перед удалением экземпляра класса
Abort()	Немедленно закрывает файл, не обращая внимания на ошибки
Close()	Закрывает файл
Duplicate()	Создает экземпляр класса для дубликата файла
Flush()	Сбрасывает данные из потока
GetFileName()	Считывает имя файла
GetFilePath()	Считывает полный путь файла
GetFileTitle()	Считывает имя файла без расширения (file title)
GetLength()	Считывает длину файла
GetPosition()	Считывает текущую позицию в файле
GetStatus()	Считывает статус файла
LockRange()	Блокирует фрагмент файла для доступа со стороны других процессов
Open()	Открывает файл
Read()	Считывает данные из файла
Remove()	Стирает файл
Rename()	Переименовывает файл
Seek()	Переставляет указатель текущей позиции в файле
SeekToBegin()	Устанавливает указатель текущей позиции на начало файла
SeekToEnd()	Устанавливает указатель текущей позиции на конец файла
SetFilePath()	Устанавливает путь к файлу
SetLength()	Устанавливает длину файла
SetStatus()	Устанавливает статус файла
UnlockRange()	Снимает блокировку фрагмента файла
Write()	Записывает данные в файл

Как видно из приведенной таблицы, набор методов класса CFile предоставляет широкие возможности для работы с файлами. В этом разделе вы найдете примеры использования лишь некоторых методов класса CFile. Однако большинство перечисленных методов просто не нуждается в дополнительных разъяснениях — их применение не вызывает трудностей у любого программиста, который когда-либо имел дело с языком С.

Ниже приведен фрагмент текста программы, в котором создается и открывается файл, в него записывается строковая переменная, а затем извлекается некоторая информация о файле.

```
// Создать файл.
CFile file("TESTFILE.TXT",
   CFile::modeCreate | CFile::modeWrite);

// Записать данные в файл.
CString message( "Hello file!");
int length = m_message.GetLength();
file.Write((LPCTSTR)m_message, length);
```

```
// Считать информацию о файле.
CString filePath = file.GetFilePath();
int fileLength = file.GetLength();
```

Обратите внимание на то, что при вызове конструктора с аргументом — именем файла — нет необходимости явно вызывать функцию открытия файла. Аргументы конструктора — имя файла и флаги режима доступа к файлу. Как и обычно, можно передавать сразу несколько флагов (если, конечно, они не противоречат друг другу), объединяя их по *ИЛИ*. Эти флаги объявлены в рамках класса CFile и определяют как режим открытия файла, так и ограничения на доступ к файлу. Флаги и их описания перечислены в табл. 7.2.

Таблица 7.2. Флаги режима доступа к файлу

Флаг	Описание
CFile::modeCreate	Создается новый файл или усекается существующий; в результате его длина становится равной 0
CFile::modeNoInharit	Запрещается наследование файла порожденным процессом
CFile::modeNoTruncate	Если файл уже создан, его содержимое не удаляется
CFile::modeRead	Файл открывается только для чтения
CFile::modeReadWrite	Файл открывается для чтения и записи
CFile::modeWrite	Файл открывается только для записи
CFile::shareCombat	Позволяет любому другому процессу открывать этот файл
CFile::shareDenyNone	Разрешает другим процессам читать и выполнять запись в файл
CFile::shareDenyRead	Запрещается чтение файла другими процессами
CFile::shareDenyWrite	Запрещает другим процессам выполнять запись в файл
CFile::shareExclusive	Запрещает доступ к файлу другим процессам
CFile::typeBinary	Устанавливает для файла двоичный режим
CFile::typeText	Устанавливает для файла текстовый режим

После создания файла формируется строковая переменная, определяется ее длина и затем переменная записывается в файл. Для этого вызывается метод Write()класса CFile. В качестве аргументов функции Write()передаются адрес буфера данных и количество байтов данных, которые подлежат записи в файл. Обратите внимание на преобразование типа аргумента функции — приведение к типу LPCTSTR. Оператор преобразования типа такого вида объявлен в классе CString и извлекает строку из экземпляра класса.

Обратите внимание еще на одну особенность этого фрагмента программы. В нем отсутствует явное обращение к методу Close(), поскольку файл автоматически закрывается деструктором локального объекта file, как только последний выходит из области видимости.

Считывание данных из файла не намного сложнее, чем запись в него:

```
// Открыть файл.
CFile file("TESTFILE.TXT", CFile::modeRead);

// Прочесть данные из файла.
char s[81];
int bytesRead = file.Read(s, 80);
s[bytesRead] = 0;
CString message = s;
```

На этот раз при открытии файла задается флаг CFile::modeRead и таким образом задается режим обращения к файлу только для чтения. После этого программа создает буфер символов и вызывает метод Read() класса CFile, который и выполняет чтение данных из файла в заданный буфер. Аргументами метода Read() являются адрес буфера-приемника и количество байтов, которые следует прочесть из файла. Метод возвращает количество действительно прочитанных байтов, которое в данном случае практически всегда будет меньше, чем запрашиваемые 80. Число считанных байтов нужно программе, чтобы добавить байт 0 в конец считанной последовательности, ограничивая таким образом ее, как строку символов. В результате формируется стандартная строковая переменная языка С, которую можно переслать в переменную message.

В этом фрагменте программы имя файла задается жестко в виде строковой константы. Для того чтобы получить имя требуемого файла от пользователя, придется немного потрудиться. Загляните в электронную документацию MFC и найдите в ней описание класса CFileDialog. Он очень прост в использовании, но позволяет без особого труда организовать запрос у пользователя имени необходимого файла.

Создание объекта класса CArchive

Работать с файлами можно, используя объекты класса CFile, но мы рекомендуем вам пойти дальше и создать собственный объект класса CArchive (будем называть его *архивом*). Его можно использовать точно так, как и объект этого же класса в функции Serialize()[8]. Это позволит использовать функции Serialize(), написанные для объектов других классов, передавая им просто ссылку на ваш собственный объект класса CArchive.

Для создания архива нужно сформировать экземпляр класса CFile и передать его конструктору класса CArchive. Например, если планируется записывать данные из объектов каких-либо классов в файл через архив, нужно создать этот архив с помощью такой последовательности операторов:

```
CFile file("FileNAME.EXT", CFile::modeWrite);

CArchive ar(&file, CArchive::store);
```

Затем можно использовать его так же, как и архивы, создаваемые MFC без явного участия программиста. Поскольку в данном примере при создании экземпляра класса CArchive использован флаг CArchive::store, любой вызов метода IsStoring()будет возвращать TRUE и, таким образом, приведет к выполнению фрагмента программы, организующего передачу данных из объекта в архив. После завершения работы с архивом соответствующий файл можно закрыть, вызвав метод Close() либо для объекта класса CFile, либо для объекта класса CArchive:

```
ar.Close();
file.Close();
```

Если же объект выходит из области существования (например, локальная переменная функции при выходе из функции), то его уничтожение выполняется компилятором автоматически и соответствующий деструктор вызывает команду закрытия файла. Таким образом, все это происходит без участия программиста. Работая с локальными архивами, можно не беспокоиться об их закрытии.

---

[8] Напомним, что функция Serialize() является членом класса документа приложения, поддерживающего механизм сохранения-восстановления, или других классов-наследников CObject. — *Прим. ред.*

Системный реестр Registry

В первых версиях операционной среды Windows приложения хранили параметры настройки и собственные опции в *файлах инициализации*, которые, как правило, имели расширение .INI. Но времена огромных файлов WIN.INI или мириад пользовательских INI-файлов канули в Лету. В современных версиях Windows приложение сохраняет необходимую для его запуска информацию в централизованном *системном реестре* Registry. И, хотя системный реестр облегчает распределение информации о настройках между несколькими процессами, жизнь программиста от этого легче не стала. В данном разделе вы сможете познакомиться с некоторыми закулисными сторонами существования системного реестра и научиться самостоятельно программировать работу с ним в приложениях.

Как устанавливается системный реестр

В прежних версиях Windows INI-файлы представляли собой обычные текстовые файлы, которые можно было обрабатывать с помощью любого текстового редактора. В отличие от них системный реестр Registry содержит и двоичную информацию, и символьную, записанную в кодах ASCII. Содержимое Registry можно редактировать только с помощью специальной программы Registry Editor или с помощью специальных функций API, разработанных именно для этой цели. Если вам когда-либо приходилось использовать Registry Editor для просмотра системного реестра, то для вас не должно быть новостью, что последний содержит огромное количество информации, организованной в виде иерархической древовидной структуры. На рис. 7.3 показано, каким образом содержимое реестра представляется пользователю при первом обращении к Registry Editor. (Эту программу, выполняемый файл которой называется RegEdit.EXE, можно найти в главной папке Windows или запустить на выполнение из меню Start (Пуск) с помощью команды Run (Выполнить), набрав в поле ввода имени файла regedit, а затем щелкнув на OK. В Windows NT файл называется RegEdit32.EXE.)

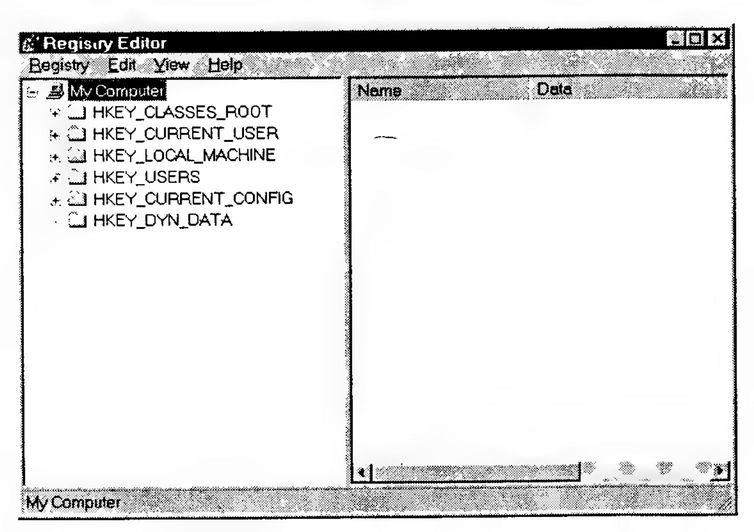


Рис. 7.3. Registry Editor позволяет просмотреть системный реестр

В левом окне перечислены предопределенные ключи реестра. Знак *плюс* рядом с именем ключа означает узел дерева, который можно "развернуть", чтобы увидеть более детальную информацию о ключе. Каждый ключ имеет несколько уровней иерархии, которые при желании можно последовательно разворачивать. Каждый ключ (или его *подключи*) может иметь (а может и НЕ иметь) ассоциированный с ним параметр. Если вы заберетесь достаточно глубоко в иерархию ключей, то в правом окне увидите перечень этих параметров. На рис. 7.4 показаны параметры, ассоциированные с текущим дизайном экрана. Если вы хотите добраться до этих параметров самостоятельно, разворачивайте последовательно HKEY_CURRENT_USER, Control Panel, Appearence и Schemes, и увидите параметры схемы дизайна экрана, установленные в системе.

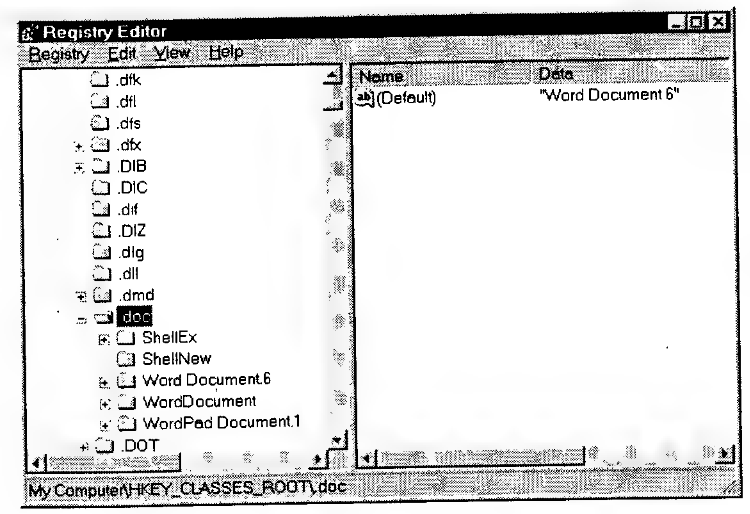


Рис. 7.4. Структура реестра представляет собой иерархическое дерево, которое содержит огромное количество информации

Предопределенные ключи

Для того чтобы узнать, что хранится в реестре, вы должны познакомиться с *предопределенными* ключами и их назначением. На рис. 7.3 представлены шесть предопределенных ключей.

- HKEY_CLASS_ROOT
- HKEY_CURRENT_USER
- HKEY_LOCAL_MACHINE
- HKEY_USERS
- HKEY_CURRENT_CONFIG
- HKEY_DYN_DATA

Ключ HKEY_CLASS_ROOT хранит информацию о типах документов и свойствах, а также информацию о классах и различных приложениях, установленных на компьютере. Например, если вы развернете этот ключ на своем компьютере, то, скорее всего, найдете в развернутом списке элемент с расширением файла .doc, под которым будут перечислены приложения, которые могут обрабатывать документы этого типа (рис. 7.5).

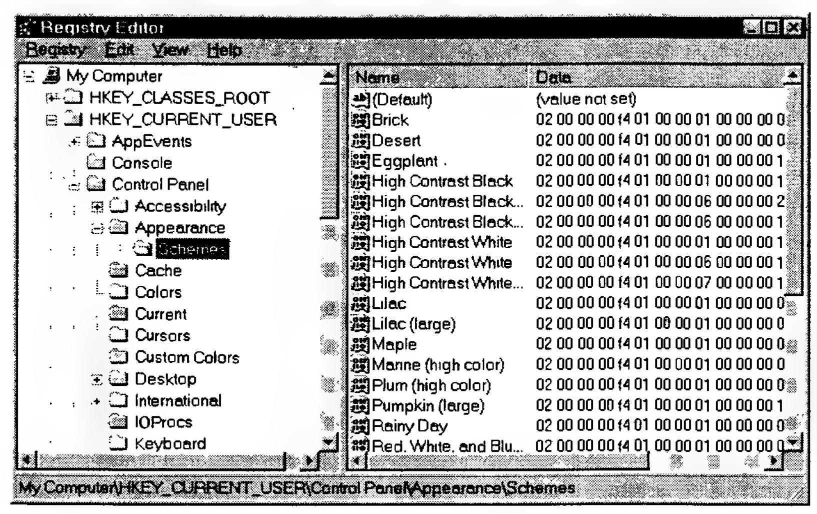


Рис. 7.5. Ключ HKEY_CLASS_ROOT хранит информацию о документах

Ключ HKEY_CURRENT_USER содержит все системные установки, сделанные пользователем, включая схему цветов, принтеры, программные группы и т.п. С другой стороны, ключ HKEY_LOCAL_MACHINE содержит статусную информацию, касающуюся компьютера, а ключ HKEY_USERS организует информацию о каждом пользователе системы и о конфигурации системы по умолчанию. И наконец, ключ HKEY_CURRENT_CONFIG содержит информацию о конфигурации аппаратных средств системы, а ключ HKEY_DYN_DATA — о динамических данных реестра, т.е. о данных, которые часто изменяются.

Использование реестра в MFC-приложениях

Теперь, когда вы кое-что узнали о реестре, позвольте мне вас несколько разочаровать — для того, чтобы описать все возможности доступа к реестру и методы его использования в приложениях, понадобится написать отдельную книгу. Как вы наверняка уже догадались, Win32 API располагает огромным множеством функций для манипуляций реестром. И если вы собрались их использовать, то, наверное, вы лучше меня знаете, как это сделать! Однако можно довольно легко добраться до реестра в MFC-приложении с тем, чтобы хранить информацию, в которой приложение нуждается во время запуска. Для того чтобы максимально упростить решение этой задачи, MFC предлагает к вашим услугам класс CWinApp и его метод SetRegistryKey(). Этот метод создает новый или открывает в дереве реестра существующий узел, соответствующий ключу вашего приложения. Все, что требуется от программиста, — передать имя ключа (как правило, имя фирмы) в качестве аргумента вызова метода, например

```
SetRegistryKey("MyCoolCompany");
```

Функцию SetRegistryKey() нужно вызвать внутри функции InitInstance() (члене класса приложения), которая вызывается только однажды — при запуске программы.

После того как будет вызвана функция SetRegistryKey(), приложение сможет создать *подключи* и задать их параметры, вызвав еще одну или две функции-члены класса CWinApp. Функция WriteProfileString() включает параметр — строку — в реестр, а WriteProfileInt() включает параметр — целое — в реестр. Для того чтобы считать параметры из реестра, можно использовать методы GetProfileString() и GetProfileInt(). (Кроме того, для обращения к параметрам реестра можно использовать методы RegSetValueEx() и RegQueryValueEx().)

На заметку

В первых версиях функции WriteProfileString(), WriteProfileInt(), GetProfileString() и GetProfileInt() программировался обмен информацией с INI–файлом. При их использовании по отдельности они по-прежнему функционируют именно так. Однако если аызову этих методоа предшествует обращение к SetRegistryKey(), то MFC переназначает приемник-источник информации — устанавливает в качестве такового системный реестр. В результате программироаание обращения к системному реестру в приложении происходит соаершенно безболезненно для программиста.

Ревизия ранее созданных приложений

В этой главе вы, возможно, сами того не подозревая, создали приложение, которое обращается к системному реестру. Ниже приведен фрагмент текста функции CMultiStringApp::InitInstance() в том виде, в котором он сгенерирован AppWizard. Точно такой же вид имеет и заготовка метода CFileDemoApp::InitInstance().

```
// Измените ключ регистрации, под которым сохраняются ваши установки.
   // После редактирования в этой строке должно быть нечто,
   // связанное с наименованием вашей компании или организации.
   SetRegistryKey(_T("Local AppWizard-Generated Applications"));

   LoadStdProfileSettings(); // Загрузка стандартных опций INI-файлов (включая MRU).
```

Аббревиатура MRU означает *Most Recently Used* (последние, которыми пользовались, — свежие) и имеет отношение к файлам, список которых приводится в меню File после того, как вы откроете с его помощью хотя бы один файл (причем не имеет значения, когда это было сделано: в текущем или предыдущих сеансах работы приложения).

ГЛАВА 8

Построение завершенного приложения ShowString

В этой главе...

Создание приложения, которое выводит на экран строку

Создание меню в приложении ShowString

Формирование диалоговых окон приложения ShowString

Как заставить работать меню

Как заставить работать диалоговое окно

Включение опций форматирования в диалоговое окно

Создание приложения, которое выводит на экран строку

В этой главе вы сможете свести воедино все приемы программирования в Visual C++, о которых шла речь в предыдущих главах. В качестве примера мы выбрали приложение, которое уже представляет определенную практическую ценность. В процессе работы над приложением вы сможете добавить в него меню, а в это меню включить дополнительные пункты, обеспечить живучесть создаваемых приложением документов и настройку формируемого изображения в соответствии с параметрами, которые вводит пользователь. В последующих главах мы будем наращивать функциональные возможности этого приложения. Надеемся, что к концу книги будет создан если не шедевр, то хотя бы программа, с которой не стыдно показаться в порядочном обществе.

Вы пройдете вместе с нами весь тернистый путь создания классического приложения для программистов, работающих на языке C, — того самого, которое выводит на экран фразу Hello, world! (Здравствуй, мир!). Приложение просто выводит в своем главном окне строку текста. *Документ*, который сохраняется в файле, содержит эту строку и несколько установок, выполненных в программе. Существует также нестандартный пункт меню, который инициирует вывод на экран диалогового окна. Пользуясь его услугами, можно изменить как текстовую строку, так и выполненные ранее установки, которые определяют параметры отображения строки. Само по себе приложение настолько просто, что включение в него нового пункта меню и диалогового окна никак не затеняется сложным алгоритмом обработки данных в приложении. Так что вызывайте на экран Visual Studio и в путь!

Создание заготовки приложения с помощью AppWizard

Первое, что нужно сделать, — это создать с помощью AppWizard заготовку приложения. (Этот материал обстоятельно изложен в главе 1.) Выберите File⇨New, затем вкладку Projects. Установите имя проекта ShowString и соответствующие каталоги для файлов проекта. При этом имена пользовательских классов должны соответствовать именам, которые вы встретите в тексте данной главы. Щелкните на OK.

На первом этапе настройки AppWizard выбор между SDI- и MDI-приложениями не имеет, по сути, никакого значения. Но вариант MDI-приложения позволит вам на собственном опыте убедиться, как просто можно переходить от приложения с единственным документом к многодокументному. А потому давайте выберем MDI-приложение. Кроме того, выберите заданный по умолчанию язык (English (United States) — американский вариант английского) и затем щелкните на Next.

В нашем приложении ShowString не предусматривается поддержка работы с базами данных или работы с составными документами. Так что можно смело щелкать на Next на втором и третьем этапах настройки AppWizard. В диалоговом окне MFC AppWizard — Step 4 выберите флажки Docking toolbar (Панель инструментов), Initial status bar (Панель состояния), Printing and print preview (Печать и предварительный просмотр распечатки), Context sensitive Help (Контекстная справка) и 3D Controls (Объемный дизайн элементов управления), а затем щелкните на Next. На следующем этапе установите включение в текст программы комментариев и использование разделяемой динамически связываемой библиотеки (DLL), после чего опять щелкните на Next. Имена классов и файлов, предлагаемые AppWizard, нас вполне устраивают, так что в окне последнего этапа настройки щелкаем на Finish. На рис. 8.1 представлено итоговое окно настройки AppWizard, в котором нужно будет подтвердить выполненную работу, щелкнув на OK.

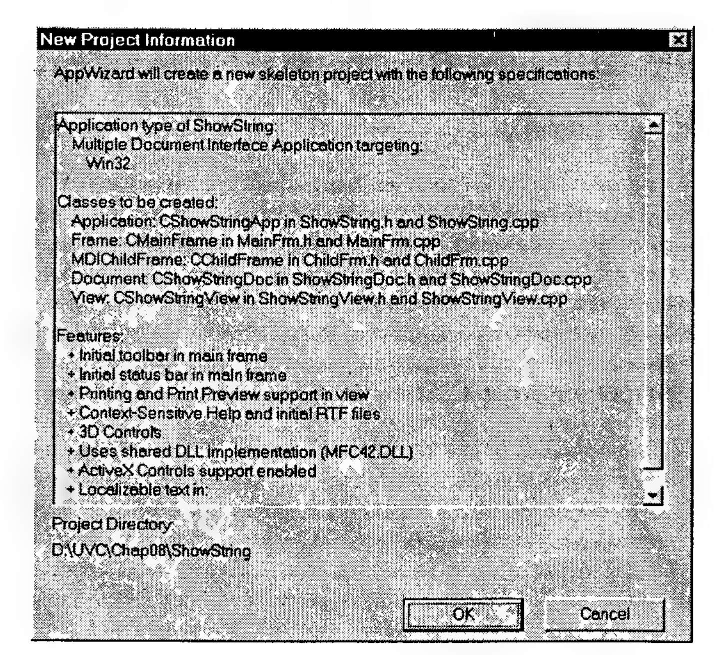


Рис. 8.1. `AppWizard` *суммирует выполненные настройки для приложения* `ShowString`

Вывод строки на экран

Приложение ShowString выводит на экран строку текста, которая включена в состав документа. Нужно добавить член-переменную в класс документа CShowStringDoc и включить операторы для ее записи и чтения в функцию Serialize(). Инициализировать строку можно в функции OnNewDocument() класса документа и перегрузить метод OnDraw() класса представления с тем, чтобы эта строка выводилась в окне приложения. О документах и их представлении речь идет в главе 4.

Члены-переменные и сохранение-восстановление

Включите объявление членов-переменных и членов-функций с помощью следующих операторов в файл ShowStringDoc.h:

```
private:
  CString string;
public:
  CString GetString() {return string;}
```

Встроенная функция позволяет другим компонентам приложения считывать строку, но не дает им возможности менять строку документа.

Далее измените заготовку функции CShowStringDoc::Serialize(), подготовленную AppWizard, таким образом, чтобы она выглядела, как в листинге 8.1. Для этого нужно развернуть класс CShowStringDoc в окне ClassView и сделать двойной щелчок на Serialize(). После этого можно приступать к редактированию текста этой функции. Поскольку вы используете MFC-класс CString, для архива уже определены терминальные операторы << и >>. Поэтому функция Serialize() в нашем варианте оказалась довольно простой. Терминальные операторы передают строку в архив при сохранении документа и считывают данные в строковые машинные переменные при загрузке документа из архива. Процедура сохранения-восстановления документа подробно описана в главе 7.

Листинг 8.1. Файл ShowStringDoc.CPP — функция CShowStringDoc::Serialize()

```
void CShowStringDoc::Serialize(CArchive& ar)
{
    if (ar.IsStoring())
    {
        ar << string;
    }
    else
    {
        ar >> string;
    }
}
```

Инициализация строки

При формировании нового документа вам понадобится инициализировать строку — установить в ней текст Hello, world!. Новый документ формируется после выбора пользователем в меню команды File⇨New. Сообщение, порожденное этой командой, перехватывается в объекте класса CShowStringApp (текст карты сообщений представлен в листинге 8.2) и обрабатывается функцией CWinApp::OnFileNew(). (Карты сообщений и обработчики сообщений подробно рассматриваются в главе 3.) В заготовках приложения, которые формируются AppWizard, функция OnFileNew() вызывается для того, чтобы создать пустой документ. В OnFileNew(), в свою очередь, вызывается метод OnNewDocument() класса документа, который и осуществляет инициализацию членов-переменных документа.

Листинг 8.2. Файл ShowString.CPP — карта сообщений

```
BEGIN_MESSAGE_MAP(CShowStringApp, CWinApp)
  //{{AFX_MSG_MAP(CShowStringApp)
  ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
  //}}AFX_MSG_MAP
  // Стандартные команды работы с файлами документа.
  ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
  ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
  // Стандартные команды работы с принтером.
  ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()
```

AppWizard предоставляет в распоряжение разработчика заготовку функции CShowStringDoc::OnNewDocument(), текст которой представлен в листинге 8.3.

Листинг 8.3. Файл ShowStringDoc.CPP — функция CShowStringDoc::OnNewDocument()

```
BOOL CShowStringDoc::OnNewDocument()
{
   if (!CDocument::OnNewDocument())
     return FALSE;

   // TODO: сюда добавить операторы реинициализации
   // (документы SDI-приложения будут повторно использовать этот документ).

   return TRUE;
}
```

Удалите комментарии и вставьте вместо них оператор:

```
string = "Hell, world!";
```

Вызов CDocument::OnNewDocument() оставьте, поскольку эта функция выполнит всю остальную черновую работу, связанную с рождением нового документа.

Вывод строки на экран

Как вы, наверное, помните из главы 5, метод OnDraw() класса представления вызывается в любой ситуации, когда необходимо обновить изображение в окне приложения — при первоначальной загрузке, изменении размеров окна, его восстановлении или когда прежде закрытое другим окно приложения выходит на авансцену. AppWizard предоставляет в ваше распоряжение заготовку этой функции, текст которой представлен в листинге 8.4. Для того чтобы ее отредактировать, разверните класс CShowStringView в окне ClassView и сделайте двойной щелчок на OnDraw().

Листинг 8.4. Файл ShowStringView.CPP — функция CShowStringView::OnDraw()

```
void CShowStringView::OnDraw(CDC* pDC)
{
  CShowStringDoc* pDoc = GetDocument();
  ASSERT_VALID(pDoc);

  // TODO: вставить сюда операторы для вывода содержимого документа.

}
```

Метод OnDraw()принимает в качестве аргумента указатель контекста устройства, как это подробно описано в главе 5. Класс контекста устройства CDC содержит метод DrawText(), который и организует вывод текста на экран. Прототип функции DrawText() имеет вид

```
int DrawText (const CString& str, LPRECT lpRect, UINT nFormat)
```

Экземпляр класса CString, который передается этой функции, есть строка из экземпляра класса документа. Доступ к ней производится через pDoc->GetString(). Аргумент lpRect — параметры прямоугольной области окна, в которой и будет выведен текст. Его можно получить, обратившись к функции GetClientRect(). И наконец, аргумент nFormat определяет формат отображения текста. Например, константа DT_CENTER задает вывод текста по центру окна относительно правой и левой границ. Константа DT_VCENTER задает центрирование относительно верхней и нижней рамок окна, но это имеет смысл только при выводе однострочного текста, который идентифицируется признаком DT_SINGLELINE. Множество флагов форматирования можно комбинировать с помощью терминального оператора. Например, комбинация DT_CENTER|DT_VCENTER|DT_SINGLELINE — это именно то, что нужно передать в качестве аргумента nFormat в нашем случае. В результате приходим к заключению, что в текст функции CShowStringView::OnDraw()нужно включить следующие операторы:

```
CRect rect;
GetClientRect(&rect);
pDC->DrawText(pDoc->GefString(), &rect, DT_CENTER|DT_VCENTER|DT_SINGLELINE);
```

Эти операторы создают экземпляр класса CRect и передают его адрес функции GetClientRect(), которая, в свою очередь, заполняет объект rect параметрами рабочей области окна. Далее функция DrawText() выводит в эту область заданный в документе текст, центрируя его по вертикали и горизонтали.

Итак, на этой стадии создания наше приложение уже способно выводить на экран текст. Конечно, негусто, но взгляните на часы. Ведь всего этого мы добились за каких-нибудь 20 минут сидения за компьютером. Оттранслируйте и скомпонуйте приложение, а затем запустите его на выполнение. Картинка на экране должна быть такой, как на рис. 8.2. Обратите внимание, что в нашем приложении есть и меню, и панель инструментов, и строка состояния, и многое другое, чем располагает каждое уважающее себя приложение Windows. А теперь, вдоволь налюбовавшись, переходите к следующему разделу — у нас еще непочатый край работы.

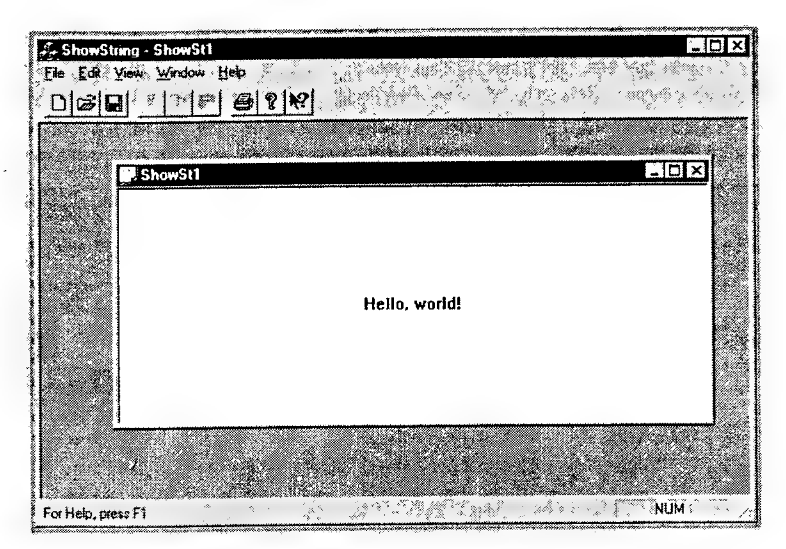


Рис. 8.2. Приложение ShowString начинает свой жизненный путь с вежливого приветствия

Создание меню в приложении ShowString

AppWizard формирует в заготовке приложения два меню, которые показаны в окне Resource View (слева на рис. 8.3). Если ни один файл не открыт в приложении, выводится меню IDR_MAINFRAME; если же хотя бы один документ открыт, выводится меню IDR_SHOWSTTYPE. Обратите внимание, что меню IDR_MAINFRAME не имеет пунктов (выпадающих меню) Edit и Window, а само меню File значительно короче, чем в IDR_SHOWSTTYPE. В первом варианте в нем есть только пункты New (Новый), Open (Открыть), Print Setup (Настройка принтера), Recent File (Свежие файлы) и Exit (Выход).

Поскольку мы собираемся добавить пункт в меню, первое, с чего нужно начать, — это выбрать, куда именно его добавлять. Нужно предоставить пользователю возможность редактировать строку, которая выведена в окне документа, и устанавливать ее формат. Можно было бы добавить пункт Value (Значение) в меню Edit (Правка). Выбор соответствующей команды должен привести к выводу на экран диалогового окна редактирования строки. Далее можно было бы создать меню Format (Формат) с единственным пунктом Appearance, выбор которого должен привести к выводу на экран другого диалогового окна, в котором можно будет настроить параметры вывода строки. Однако мы поступим по-другому — объединим обе функции в одном диалоговом окне, добавив в него соответствующие опции. Вызов этого окна поместим в новое меню Tools (Средства), в его пункт Options (Опции).

На заметку Если вы достаточно много работаете с новейшими Windows-приложениями, то наверняка обратили внимание на то, что в большинстве из них настройки помещаются в меню Tools⇨Options.

Нужно ли добавлять эти пункты в оба меню? Очевидно, нет. Если в приложении не открыт ни один документ, то негде сохранять результаты редактирования, выполненного с помощью диалогового окна. Поэтому включать новые пункты целесообразно только в меню IDR_SHOWSTTYPE. Выведите на экран это меню, сделав двойной щелчок на IDR_SHOWSTTYPE в окне ResourceView. Справа, после пункта Help (Справка), есть свободное место для нового пункта.

Щелкните на нем и наберите &Tools. Появится диалоговое окно Properties. "Пришпильте" его к рабочему столу, щелкнув на изображении канцелярской кнопки в левом углу. В поле Caption (Надпись) уже будет стоять значение &Tools. Это означает, что правое меню будет называться Tools, причем в текущий момент у него есть заготовка для одного пункта. Кроме того, справа от Tools появилось свободное место для нового меню, что видно на рис. 8.4.

Рис. 8.3. AppWizard формирует в заготовке приложения два меню



Знак & в поле Caption предшествует символу, который служит для выбора пункта меню с клавиатуры. В нашем случае это <Alt+T> для пункта Tools. В изображении названия пункта меню соответствующая литера будет подчеркнута. Для настройки обращения к меню с клавиатуры никаких усилий со стороны программиста, кроме установки символа & в нужном месте названия пункта меню, не требуется. Вы можете поэкспериментировать с разными символами ускоренного вызова меню, просто перемещая этот символ по названию пункта в поле Caption. Например, вместо &Tools введите T&ools. Тогда обращаться к этому меню можно будет, нажав комбинацию <Alt+O>.

Щелкните на изображении нового меню Tools и разместите его между Window и View. Теперь расположение пунктов меню будет напоминать "фирменные" продукты наподобие Visual Studio и Microsoft Word. Далее щелкните на пустом подпункте меню Tools. Диалоговое окно Properties изменится, и в нем отобразятся пустые поля, соответствующие еще не созданному новому подпункту. Напечатайте в поле Caption текст &Options и введите текст подсказки в поле Prompt, как это показано на рис. 8.5.

Каждый пункт меню имеет свой идентификатор, посредством которого программа связывается с ним. Visual Studio самостоятельно установит соответствующее наименование, но сейчас оно еще не будет отображено в диалоговом окне Properties. Щелкните на каком-либо другом пункте меню, а затем снова на Options. Теперь в поле ID появится ID_TOOLS_OPTIONS. Другой вариант — нажмите <Enter> после ввода наименования пункта. Тогда отметка сдви-

нется вниз — ниже установленного пункта Options — и Visual Studio приготовится сформировать еще один пункт в этом меню. Однако вы сдвиньте отметку снова на пункт Options, воспользовавшись клавишами управления курсором. Эффект будет тот же, что и в предыдущем случае — в поле ID диалогового окна Properties появится ID_TOOLS_OPTIONS.

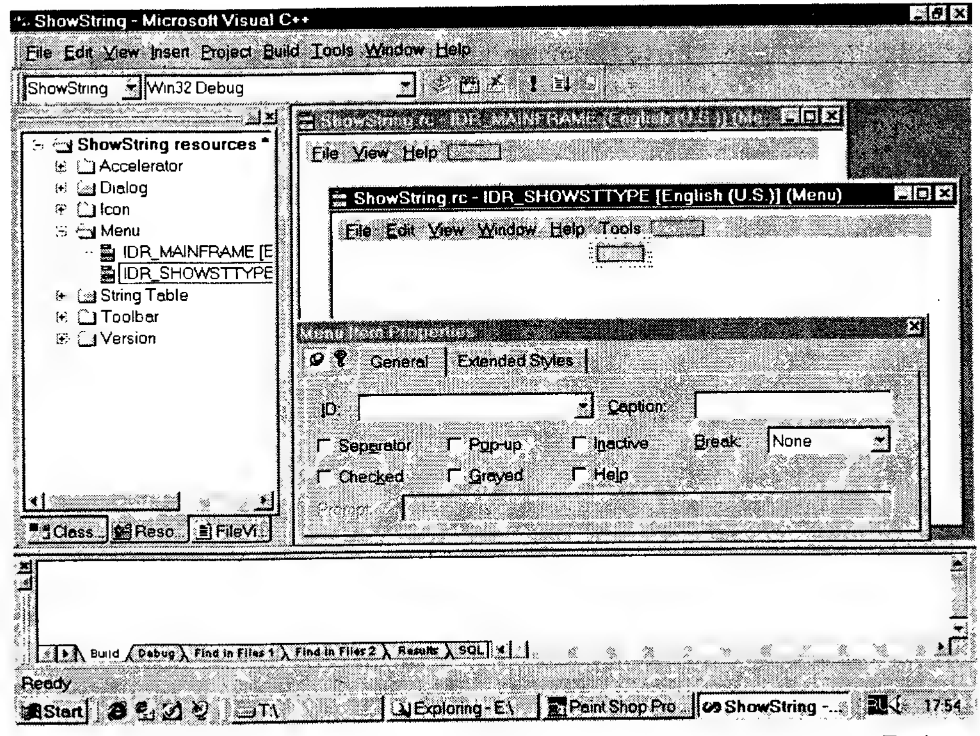


Рис. 8.4. Манипулируя в окне ResourceView, можно очень просто добавить меню Tools

Если вы считаете нужным использовать в приложении клавиши-акселераторы, например <Ctrl+C> для команды Edit⇨Copy, выполните следующее. Щелкните на значке + узла Accelerator в окне ResourceView, а затем сделайте двойной щелчок на IDR_MAINFRAME — единственной таблице акселераторов в этом приложении. Через секунду вы увидите, какие комбинации клавиш задействованы в приложении. Комбинация <Ctrl+O> уже занята, а вот <Ctrl+T> свободна. Для того чтобы подключить комбинацию <Ctrl+T> к команде Tools⇨Options, выполните следующее.

1. Щелкните на пустой строке в самом низу таблицы в окне Accelerator. Если вы уже закрыли окно Properties, верните его на экран. Для этого нужно выбрать в меню среды разработки команду View⇨Properties, а затем "пришпилить" появившееся окно к рабочему столу, щелкнув на изображении канцелярской кнопки в левом углу. Другой вариант — *дважды* щелкнуть на пустой строке в самом низу таблицы. На экране появится диалоговое окно Properties.

2. В раскрывающемся списке ID выберите ID_TOOLS_OPTIONS. Все элементы в списке отсортированы по алфавиту (естественно, латинскому). (Учтите, что список достаточно длинный, а нужный нам элемент находится в самом низу. Так что в поле ввода списка наберите ID_TO — и отметка в списке установится на нужном нам элементе.)

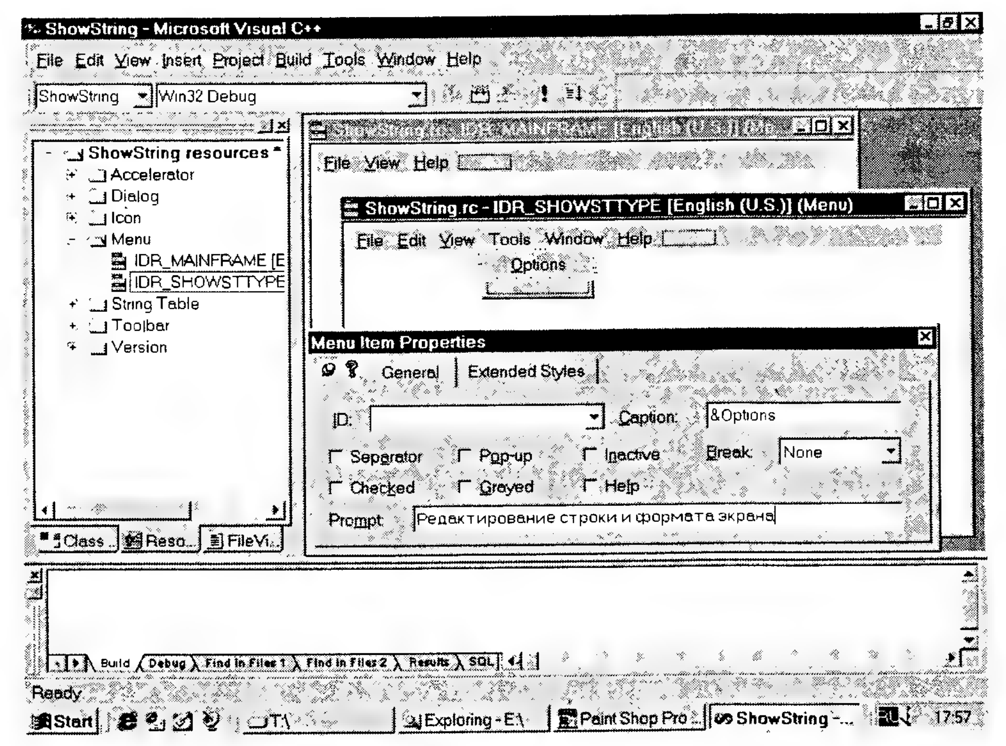


Рис. 8.5. Команда Tools⇨Options *будет управлять всеми манипуляциями строкой в приложении* ShowString

3. Введите T в поле Key. Проверьте. чтобы был установлен флажок Ctrl в группе Modifiers (Модификаторы), а флажки Alt и Shift были сброшены. Другой вариант, который приведет к этому же результату, — щелкнуть на кнопке Next Key Typed и затем набрать Ctrl+T. Все поля диалогового окна соответственно заполнятся.

4. Щелкните на другой строке таблицы Accelerator — и все введенные настройки будут зафиксированы Visual Studio.

На рис. 8.6 показано окно Properties после повторного щелчка на только что введенной строке таблицы акселераторов.

Вы еще не забыли, ради чего мы все это делаем? После того как пользователь выберет команду Tools⇨Options, должно появиться диалоговое окно. Но прежде, чем заняться подключением диалогового окна к ресурсу, нужно его (диалоговое окно) сформировать.

Формирование диалоговых окон приложения ShowString

В главе 2 приводились некоторые сведения о диалоговых окнах. В данном разделе эта тема будет продолжена. Приложение ShowString будет выводить на экран два диалоговых окна: одно по команде Tools⇨Options, а второе — диалоговое окно About. Последнее формируется практически без нашего участия самим AppWizard, но его придется немного изменить. Что касается диалогового окна Options, то его придется формировать с самого начала самостоятельно, т.е. пользуясь средствами Visual Studio.

Рис. 8.6. Комбинация клавиш — акселератор подключен к идентификатору ресурса

Диалоговое окно About приложения ShowString

На рис. 8.7 показано диалоговое окно About, которое формируется AppWizard. Оно содержит имя приложения и текущий год. Чтобы вывести диалоговое окно About нашего приложения на экран, щелкните на вкладке ResourceView в окне компонентов проекта (правое окно рабочего экрана Visual Studio), разверните узел Dialog (для этого щелкните на значке + возле надписи Dialog). После двойного щелчка на IDD_ABOUTBOX появится соответствующий ресурс — диалоговое окно About.

Но, возможно, из чувства глубокого уважения вы захотите включить в окно еще и название своей компании. Ниже в качестве примера мы опишем, как ввести в диалоговое окно название *Que Books*. Щелкните на строке текста Copyright ® 1998 в диалоговом окне, и вокруг него появится рамка выбора. Вызовите на экран окно Properties, если его там еще нет. Отредактируйте надпись в поле Caption — введите Que Books в конец текста. Это изменение немедленно появится и в окне About.



Если в вашем окне не появятся линейки, как на рис. 8.7, их нужно будет включить, выбрав Layout⇨Guide Settings, а затем выбрав опции Rulers и Guides в верхней части диалогового окна Guide Settings.

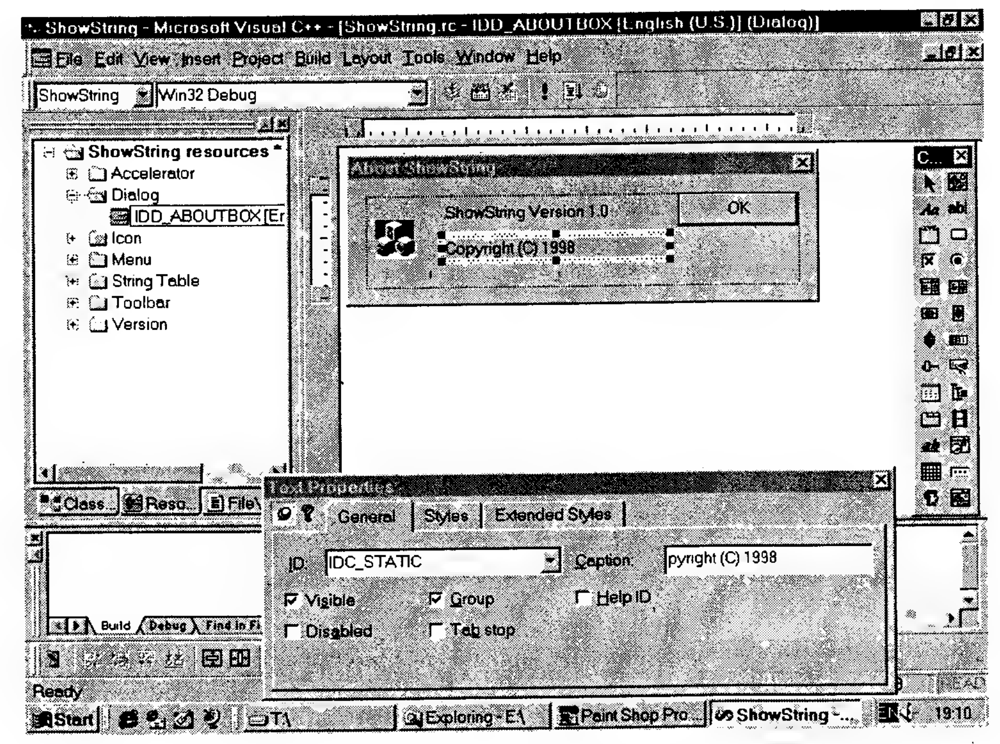


Рис. 8.7. AppWizard самостоятельно формирует диалоговое окно About

Можно сделать еще одно добавление — напомнить пользователям, из какой книги взято это приложение. Вот что для этого нужно сделать.

1. Несколько увеличьте высоту окна. Для этого сначала выберите все диалоговое окно, просто щелкнув на нем. Затем щелкните на размерном маркере в середине нижней полки окна и "оттяните" ее слегка вниз. (Такая технология редактирования графических объектов приложения и дала название программному продукту — Visual C++.)

2. В наборе элементов управления Controls (Инструментарии) щелкните на пиктограмме с надписью Aa. Таким образом вы выберете элемент управления типа static control (статическая надпись). Это просто небольшой текст, который не может быть изменен пользователем непосредственно, т.е. идеальное средство для ввода различных надписей в диалоговом окне. Щелкните теперь на поле диалогового окна в том месте, куда собираетесь поместить надпись, — где-нибудь ниже уже существующих.

3. В окне Properties измените надпись в поле Caption — вместо Static наберите Visual C++ 6. Окно автоматически увеличится в размерах, чтобы уместить введенный текст.

4. Удерживая нажатой клавишу <Ctrl>, щелкните на двух других текстовых строках в окне About. После этого выберите в меню среды разработки Layout⇨Align Controls⇨Left (Компоновка⇨Подравнять⇨Слева). В результате выбранные надписи подравняются по левому краю, причем тот элемент управления, который был выбран последним, останется на месте, а остальные будут равняться по нему.

5. Теперь — последний штрих. Выберите Layout⇨Space Evenly⇨Down (Компоновка⇨Равный интервал⇨Вниз). Будет выровнен интервал между надписями. Нужно отметить, что команды меню Layout при правильном использовании экономят время разработчика — избавляют его от необходимости вручную изменять положение элементов управления на поле диалогового окна.

Диалоговое окно About в результате всех этих манипуляций должно приобрести вид, представленный на рис. 8.8.

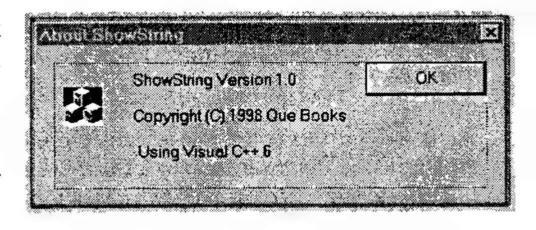


Рис. 8.8. В течение считанных минут можно изменить диалоговое окно About *по своему вкусу*



Все пункты меню Layout продублированы пиктограммами на панели инструментов Dialog.

Диалоговое окно Options приложения ShowString

Теперь вас не затруднит создание диалогового окна Options. Сначала сформируйте заготовку нового окна, выбрав Insert⇨Resource (Вставить⇨Ресурс), а затем сделав двойной щелчок на Dialog. Появится пустое диалоговое окно, которому по умолчанию будет присвоено наименование Dialog1. В этом окне уже будут установлены кнопки OK и Cancel, как четко видно на рис. 8.9.

Рис. 8.9. В новом диалоговом окне всегда установлены кнопки OK *и* Cancel

А теперь, если вы будете следовать нашим инструкциям, это пустое окно превратится в средство редактирования строки приложения.

1. В поле ID введите IDD_OPTIONS, а в поле Caption — Options.
2. В наборе элементов управления Controls (Инструментарии) щелкните на пиктограмме с надписью ab|. При этом будет выбран элемент управления типа edit box (текстовое поле). Его можно использовать для ввода и редактирования нового текста для строки документа приложения. Щелкните на поле проектируемого диалогового окна в том месте, где предполагается установить текстовое поле. Задайте идентификатор этого элемента управления — в поле ID введите IDC_OPTIONS_STRING. Напоминаем, что идентификаторы всех элементов управления согласно принятому соглашению об именах должны начинаться с IDC_, а затем включать имя диалогового окна и собственно элемента управления.
3. Растяните текстовое поле как можно больше по длине, используя для этого размерные маркеры рамки выбора.
4. Добавьте еще один элемент управления типа static control (статическая надпись) чуть выше текстового поля и введите для него в поле Caption текст надписи — String.

Мы еще раз вернемся к этому диалоговому окну позже, когда будем добавлять элементы управления, необходимые для настройки формата строки. Но сейчас наше диалоговое окно уже вполне готово к подсоединению к остальной программе. Выглядеть оно должно так, как на рис. 8.10.



Рис. 8.10. Диалоговое окно Options предназначено для редактирования строки документа приложения

Как заставить работать меню

Диалоговое окно Options должно появиться на экране после выбора пользователем команды меню Tools⇨Options. Для того чтобы организовать вызов одной из функций при выборе этой команды, призовем на помощь ClassWizard. Затем останется ерунда — написать текст программы этой функции, в которой будет создан экземпляр класса диалогового окна, и вывести его изображение на экран.

Класс диалогового окна

Собственно заготовку класса диалогового окна создаст для вас ClassWizard. В то время как фокус ввода находится в окне, которое отображает диалоговое окно IDD_OPTIONS, выберите в меню Visual Studio View⇨ClassWizard. ClassWizard обнаружит, что класса, соответствующего этому диалоговому окну, еще не существует, и предложит его создать (рис. 8.11).

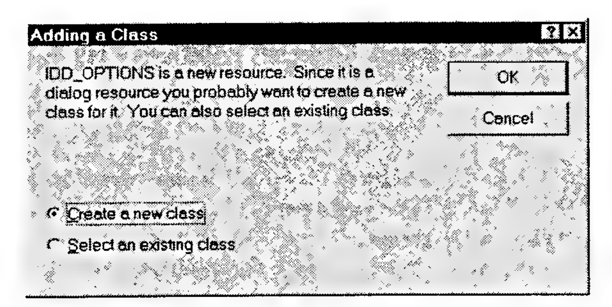


Рис. 8.11. Создание класса C++ для нового диалогового окна

Не изменяйте установку переключателя Create a new class и щелкните на OK. Появится диалоговое окно Create New Class, представленное на рис. 8.12.

Заполните поля диалогового окна следующим образом.

1. Выберите осмысленное имя для нового класса. Оно должно начинаться с `C` и включать слово `Dialog`. Например, выберем `COptionsDialog`.

   По умолчанию в качестве базового класса предлагается выбрать `CDialog`, что нам вполне подходит.

2. Щелкните на OK — и класс будет создан.

До сих пор `ClassWizard` поджидал своей очереди и не показывался на глаза, скрываясь за другими окнами на экране. Теперь наступил его черед. Щелкните на вкладке Member Variables и свяжите `IDC_OPTIONS_STRING` с переменной `m_string` — экземпляром класса `CString`. Эту процедуру мы уже выполняли в главе 2. Теперь щелкните на OK и закройте `ClassWizard`.

Вам, конечно же, интересно, какой текст программы сформировал `ClassWizard` для объявления нового класса. Текст нового файла заголовка приведен в листинге 8.5.

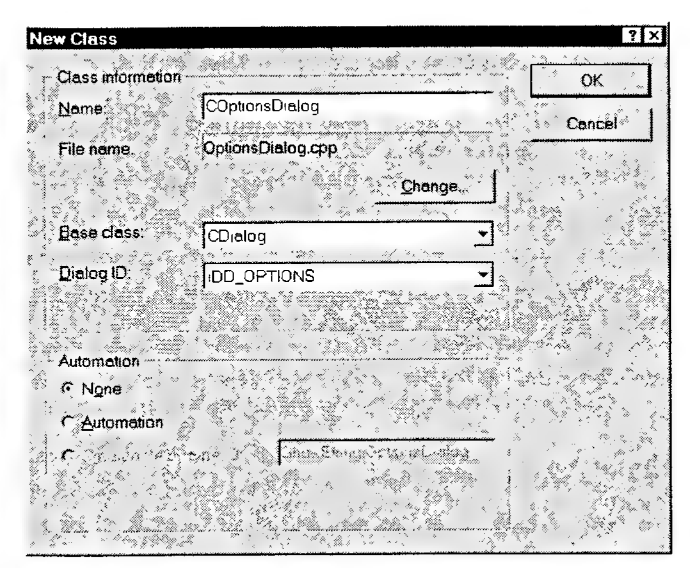


Рис. 8.12. Класс нового диалогового окна является наследником класса `CDialog`

Листинг 8.5. Файл OptionsDialog.H — файл заголовка для класса COptionsDialog

```
// OptionsDialog.h : файл заголовка.
//

//////////////////////////////////////////////////////////////////////////////
// Диалоговое окно COptionsDialog.

class COptionsDialog : public CDialog
{
// Конструкторы.
public:
  COptionsDialog(CWnd* pParent = NULL);   // Стандартный конструктор.

// Данные для диалогового окна.
  //{{AFX_DATA(COptionsDialog)
  enum { IDD = IDD_OPTIONS };
  CString  m_string;
  //}}AFX_DATA
// Перегрузка.
  // ClassWizard генерирует перегрузки виртуальных функций
  //{{AFX_VIRTUAL(COptionsDialog)
  protected:
  virtual void DoDataExchange(CDataExchange* pDX);   // Поддержка DDX/DDV.
  //}}AFX_VIRTUAL

// Реализация.
protected:

  // Сформированные функции карты сообщений.
  //{{AFX_MSG(COptionsDialog)
    // ВНИМАНИЕ!!  Здесь ClassWizard будет добавлять функции-члены.
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()
};
```

Обратите внимание на огромное количество замысловатых комментариев. Большинство из них предназначено не для программиста, а для самого ClassWizard. Когда придет время работать с этим файлом, ClassWizard будет ориентироваться по этим комментариям. Из полезных вещей здесь есть одна переменная-член (m_string), объявлены один конструктор и одна функция-член (DoDataExchange()), которая извлекает данные из члена-переменной и, наоборот, устанавливает ее значение. Файл текста программы не намного длиннее. Он представлен в листинге 8.6.

Листинг 8.6. Файл OptionsDialog.CPP — файл реализации для класса COptionsDialog

```
// OptionsDialog.cpp: файл реализации.
//

#include "stdafx.h"
#include "ShowString.h"
#include "OptionsDialog.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

```
/////////////////////////////////////////////////////////////////////////////
// Диалоговое окно COptionsDialog.

COptionsDialog::COptionsDialog(CWnd* pParent /*=NULL*/)
  : CDialog(COptionsDialog::IDD, pParent)
{
  //{{AFX_DATA_INIT(COptionsDialog)
  m_string = _T("");
  //}}AFX_DATA_INIT
}

void COptionsDialog::DoDataExchange(CDataExchange* pDX)
{
  CDialog::DoDataExchange(pDX);
  //{{AFX_DATA_MAP(COptionsDialog)
  DDX_Text(pDX, IDC_OPTIONS_STRING, m_string);
  //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(COptionsDialog, CDialog)
  //{{AFX_MSG_MAP(COptionsDialog)
    // ВНИМАНИЕ: сюда ClassWizard будет вставлять макросы карты сообщений.
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

Конструктор очищает строковую переменную-член. Соответствующий оператор выделен с двух сторон специальными комментариями, которые в будущем помогут ClassWizard найти место в файле, чтобы инициализировать новые переменные-члены класса, если они будут включены в состав класса. Функция DoDataExchange() вызывает DDX_Text() для передачи данных из элемента управления, у которого идентификатор ресурса IDC_OPTIONS_STRING, в переменную-член m_string и наоборот. Этот оператор также выделен специальными комментариями. И наконец, здесь есть пустая карта сообщений, поскольку на этом этапе работы над программой еще не организован перехват сообщений классом COptionsDialog.

Перехват сообщений

Следующий этап создания приложения ShowString — организация перехвата сообщения, которое посылается после выбора пользователем команды Tools⇨Options. В приложении ShowString используется семь классов: COptionsDialog, CAboutDlg, CChildFrame, CMainFrame, CShowStringApp, CShowStringDoc и CShowStringView. Какой из них должен перехватывать команду? Строка и опции форматирования должны сохраняться в документе, а использоваться при выводе на экран в представлении. Так что перехватывать команду должен один из двух классов, ответственных за эти функции. Закрытые члены-переменные — это члены класса документа, который не позволит их изменить с помощью методов класса представления до тех пор, пока не будут сформированы открытые (public) методы установки значения строки. Так что наиболее целесообразно передать функции перехвата классу документа.

Для того чтобы организовать такой перехват, выполните следующее.

1. Выведите на экран ClassWizard, если он еще не вызван.
2. Щелкните на вкладке Message Maps.
3. Выберите CShowStringDoc из раскрывающегося списка ClassName.
4. Выберите ID_TOOLS_OPTIONS из раскрывающегося списка ObjectIDs в левой части экрана и COMMAND из списка Messages в правой.
5. Щелкните на Add Function, чтобы включить функцию, которая будет обрабатывать эту команду.

6. Появится диалоговое окно Add Member Function, показанное на рис. 8.13. В нем можно отредактировать имя функции обработки, предлагаемое ClassWizard. Я бы вам этого не советовал, а потому щелкните на OK в знак согласия.

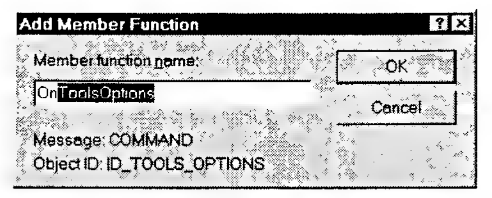


Рис. 8.13. ClassWizard предлагает имя функции, которая будет обрабатывать сообщение



Не рекомендуется менять имена функций, которые предлагает ClassWizard. Если же вы все-таки сочтете нужным это сделать (возможно, предлагаемое имя будет слишком длинным или его будет неприлично произнести в обществе), то свой вариант имени функции начинайте с On. В противном случае тому, кому выпадет честь сопровождать разработанную программу в дальнейшем, будет не так просто найти в листинге эту функцию обработки сообщения.

Теперь щелкните на Edit Code — ClassWizard закроется и можно будет приступить к редактированию текста только что созданной функции. Что же произошло с классом CShowStringDoc после того, как вы организовали перехват сообщения ID_TOOLS_OPTIONS? Новая карта сообщений в соответствующем файле заголовка представлена в листинге 8.7

Листинг 8.7. Файл ShowStringDoc.H — карта сообщений для класса CShowStringDoc

```
// Функции, сгенерированные для карты загрузки.
protected:
   //{{AFX_MSG(CShowStringDoc)
   afx_msg void OnToolsOptions();
   //}}AFX_MSG
   DECLARE_MESSAGE_MAP()
```

Это только объявление функции. В файле же текстов программ ClassWizard изменил карту сообщений, и она стала выглядеть так, как представлено в листинге 8.8.

Листинг 8.8. Файл ShowStringDoc.CPP — карта сообщений для класса CShowStringDoc

```
// Функции, сгенерированные для карты загрузки.
protected:
   //{{AFX_MSG(CShowStringDoc)
   afx_msg void OnToolsOptions();
   //}}AFX_MSG
   DECLARE_MESSAGE_MAP()
```

Такая карта сообщений организует вызов функции OnToolsOptions()в ответ на команду ID_TOOLS_OPTIONS. Кроме того, ClassWizard добавил заготовку для функции OnToolsOptions():

```
void CShowStringDoc::OnToolsOptions()
{
   //TODO: сюда добавьте операторы обработки сообщения.
}
```

Как заставить работать диалоговое окно

Метод OnToolsOptions() должен инициализировать и вывести на экран диалоговое окно, а затем что-то сделать с теми данными, которые введет пользователь. Этот процесс мы обсуждали в главе 2. Вы уже связали элемент управления — текстовое поле — с переменной m_string — членом класса диалогового окна. Нужно инициализировать эту переменную перед выводом диалогового окна, а затем считать те данные, которые занесет в нее пользователь.

Текст метода OnToolsOptions()представлен в листинге 8.9. Эта функция выводит на экран диалоговое окно и принимает от него данные. Введите текст в заготовку, которую сформировал ClassWizard.

Листинг 8.9. Файл ShowStringDoc.CPP — метод OnToolsOptions()

```
void CShowStringDoc::OnToolsOptions()
{
   COptionsDialog dlg;
   dlg.m_string = string;

   if (dlg.DoModal() == IDOK)
   {
      string = dlg.m_string;
      SetModifiedFlag();
      UpdateAllViews(NULL);
   }

}
```

Первым делом, эта функция дублирует данные из переменной-члена класса документа в переменную-член класса диалогового окна. ClassWizard объявил m_string как открытый (public) член класса COptionsDialog, так что для класса документа доступ к ней открыт. Диалоговое окно выводится на экран обращением к его методу DoModal(). Если пользователь завершит работу с диалоговым окном, щелкнув на OK, полученные от него данные будут переписаны в переменную-член объекта класса документа, будет установлен флаг изменения документа (а это значит, что перед закрытием приложения пользователю будет предложено сохранить измененный документ) и вызов функции UpdateAllViews() обеспечит обновление изображения в окне документа. Для того чтобы компилятор смог скомпилировать этот текст, добавьте в самом начале строку директивы для компилятора:

```
#include "OptonsDialog.h"
```

Теперь можно снова оттранслировать и скомпоновать приложение, а затем его запустить. После запуска выберите в меню приложения Tools⇨Options и отредактируйте строку документа. Щелкните на OK в диалоговом окне редактирования — и увидите результат своих титанических усилий в окне документа. Попытайтесь закрыть приложение — программа предложит вам сохранить файл. Так и поступите, а затем вновь запустите приложение. Вновь откройте файл — документ с текстом по умолчанию “Hello, world!” остается открытым, но, кроме него, откроется новый документ с отредактированным текстом. Итак, приложение работает, в чем вы можете удостовериться, взглянув на рис. 8.14. (Окна сдвинуты одно относительно другого специально, чтобы их можно было видеть на рисунке.)

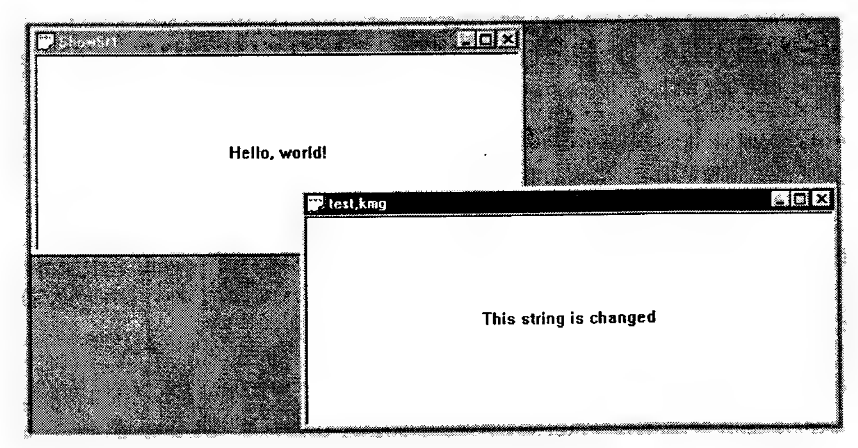


Рис. 8.14. Приложение `ShowString` *теперь может изменять строку в документе, сохранять ее в файле, а затем вновь загружать*

Включение опций форматирования в диалоговое окно

В общем, забот у приложения `ShowString` не очень много — оно может нам продемонстрировать работу с меню и диалоговым окном, причем у последнего единственным полезным элементом является текстовое поле. В этом разделе мы прибавим ему работы — включим в диалоговое окно несколько переключателей, которые будут определять опции форматирования строки.

Модификация диалогового окна Options

Не представляет особого труда включить в наше приложение стандартное диалоговое окно настройки шрифтов Font, обладающее достаточно разнообразными возможностями. Но мы поступим еще проще — установим в нашем родном диалоговом окне Options несколько переключателей: переключатель, с помощью которого пользователь сможет выбрать один из предлагаемых цветов, переключатель-флажок, который позволит установить горизонтальное центрирование строки, и переключатель-флажок для центрирования строки по вертикали. Поскольку это независимые переключатели, пользователь может выбрать либо один, либо другой, либо оба.

Откройте диалоговое окно `IDD_OPTIONS`. Для этого дважды щелкните на соответствующем элементе в окне ResourceView. Теперь добавьте в него переключатели, для чего выполните следующее.

1. Увеличьте высоту диалогового окна, чтобы появилось место для группы новых переключателей.
2. В наборе элементов управления Controls (Инструментарии) щелкните на пиктограмме с изображением переключателя (radio button). Щелкните на поле диалогового окна в том месте, где предполагается установить выбранный элемент.
3. Верните на экран окно Properties. Для этого нужно выбрать в меню среды разработки команду View⇨Properties, а затем "пришпилить" появившееся окно к рабочему столу, щелкнув на изображении канцелярской кнопки в левом углу.
4. Задайте идентификатор нового элемента управления — первого переключателя. В поле ID введите `IDC_OPTIONS_BLACK`, а в поле Caption — `&Black`.

5. Установите флажок Group; так вы укажете, что данный переключатель — первый в группе аналогичных.
6. Добавьте еще один переключатель, для которого установите идентификатор ресурса IDC_OPTIONS_RED, а надпись — &Red. Не устанавливайте флажок Group, поскольку новый элемент входит в ту же группу, что и предыдущий.
7. Добавьте третий переключатель, для которого установите идентификатор ресурса IDC_OPTIONS_GREEN, а надпись — &Green. И опять не трогайте флажок Group.
8. Переместите переключатели на поле формы диалогового окна таким образом, чтобы они оказались примерно на одной горизонтали, а затем заключите их в рамку выбора, чтобы выровнять.
9. Выберите в меню среды разработки Layout⇨Align Controls⇨Bottom. В результате выбранные переключатели выровняются в ряд по горизонтали.
10. Выберите Layout⇨Space Evenly⇨Across — и будет выровнен интервал между элементами.

Теперь аналогичным образом установим в диалоговом окне флажки.

1. В наборе элементов управления Controls щелкните на пиктограмме с изображением флажка (check box). Щелкните на поле диалогового окна в том месте, где предполагается установить выбранный элемент.
2. Задайте идентификатор нового элемента управления. В поле ID введите IDC_OPTIONS_HORIZCENTER, а в поле Caption — Center &Horizontally.
3. Щелкните на флажке Group; так вы укажете, что закончилась группа переключателей.
4. Добавьте еще один флажок, для которого установите идентификатор ресурса IDC_OPTIONS_VERTCENTER, а надпись — Center &Vertically.
5. Скомпонуйте пару новых флажков на поле диалогового окна сразу под группой переключателей.
6. В наборе элементов управления Controls щелкните на пиктограмме Group (Группа), затем очертите мышью группу переключателей на поле диалогового окна. Установите для группы название — в поле Caption введите Text Color.
7. Кнопки OK и Cancel сместите в нижнюю часть диалогового окна.
8. Выберите по очереди каждую горизонтальную группу элементов управления и выполните команду Layout⇨Center in Dialog⇨Horizontal. Элементы скомпонуются поближе к центру окна.
9. Выберите Edit⇨Select All, а затем перетащите всю рамку выбора (в нее будут теперь входить все элементы управления, включенные в состав диалогового окна) поближе к верхней полке. Сожмите рамку окна таким образом, чтобы удалить лишнее пустое пространство. Полученный шедевр компьютерного дизайна должен походить на то, что показано на рис. 8.15.

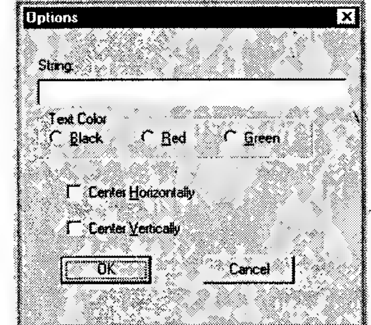


Рис. 8.15. Новая компоновка диалогового окна Options приложения ShowString



Если вы не совсем уверенно ориентируетесь в пиктограммах инструментария элементов управления, вспомните о контекстном окне указателя. Стоит вам на несколько секунд задержать указатель мыши на выбранной пиктограмме, как откроется это окно, а в нем — наименование соответствующего элемента управления.

И последний штрих — установка последовательности перехода фокуса ввода между элементами управления при использовании клавиши <Tab>. Для этого выберите в меню команду Layout⇨Tab Order, а затем щелкайте на элементах в следующем порядке.

1. IDC_OPTIONS_STRING
2. IDC_OPTIONS_BLACK
3. IDC_OPTIONS_RED
4. IDC_OPTIONS_GREEN
5. IDC_OPTIONS_HORIZCENTER
6. IDC_OPTIONS_VERTCENTER
7. IDOK
8. IDCANCEL

Затем щелкните где-нибудь вне поля диалогового окна, чтобы надписям — статическим текстовым полям — фокус ввода никогда не достался.

Включение членов-переменных в класс диалогового окна

Если уж вы добавили элементы управления в диалоговое окно, не мешает позаботиться и о включении соответствующих переменных в класс COptionsDialog в качестве полноправных членов. Вызовите ClassWizard, выберите вкладку Member Variables (Члены-переменные) и включите в класс переменные для каждого нового элемента управления. На рис. 8.16 показан результат вашей работы: список новых членов-переменных класса COptionsDialog. Флажки связаны с переменными типа BOOL; они принимают значение TRUE, если флажок установлен, и FALSE — в противном случае. Несколько иначе обстоит дело с переключателями. Только один из них (первый, тот, от которого начался отсчет группы) связан с переменной. Она имеет тип int и представляет собой индекс выбранного переключателя внутри группы (отсчет начинается с 0). Другими словами, если выбран переключатель Black, переменная m_color получит значение 0, если выбран Red — m_color получит значение 1, а Green — значение 2.

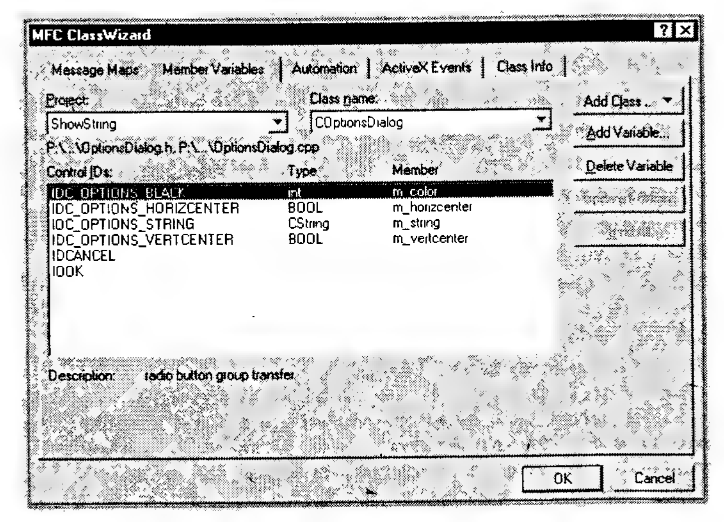


Рис. 8.16. Переменные-члены класса диалогового окна связываются с отдельными элементами управления или с группой зависимых переключателей

Включение новых переменных в документ

Переменные, которые мы собираемся включить в документ, — те же самые, что включены в диалоговое окно. Их нужно добавить в определение класса CShowStringDoc в файле заголовка, а также в тексты функций OnNewDocument() и Serialize(). Операторы, приведенные в листинге 8.10, нужно вставить в самом начале определения класса CShowStringDoc в файле ShowStringDoc.h, заменив объявление string и GetString().

Листинг 8.10. Файл ShowStringDoc.H: объявление членов-переменных класса CShowStringDoc

```
private:
   CString string;
   int   color;
   BOOL horizcenter;
   BOOL vertcenter;
public:
   CString GetString() {return string;}
   int   GetColor() {return color;}
   BOOL GetHorizcenter() {return horizcenter;}
   BOOL GetVertcenter() {return vertcenter;}
```

Что касается string, то в объявлении присутствует закрытая переменная (private) и открытая функция считывания GetString(), но нет функции установки значения этой переменной. Все эти переменные следует включить и в метод Serialize(), как это представлено в листинге 8.11. Чтобы отредактировать текст функции, сделайте двойной щелчок на ее имени в окне ClassView и принимайтесь за дело.

Листинг 8.11. Файл ShowStringDoc.CPP — метод Serialize()

```
void CShowStringDoc::Serialize(CArchive& ar)
{
   if (ar.IsStoring())
   {
      ar << string;
      ar << color;
      ar << horizcenter;
      ar << vertcenter;
   }
   else
   {
      ar >> string;
      ar >> color;
      ar >> horizcenter;
      ar >> vertcenter;
   }
}
```

И наконец, нужно инициализировать эти переменные в функции OnNewDocument(). Что выбрать в качестве значений по умолчанию для новых переменных? Как принято выражаться в определенных кругах, "вопрос неоднозначный"! Мы предлагаем сохранить строгий пуританский стиль, который так любят в старой доброй Англии, — черный текст, центрированный как по вертикали, так и по горизонтали. Новый текст функции OnNewDocument(), обеспечивающий такие установки, приведен в листинге 8.12.

Листинг 8.12. Файл ShowStringDoc.CPP — метод OnNewDocument()

```
BOOL CShowStringDoc::OnNewDocument()
{
  if (!CDocument::OnNewDocument())
    return FALSE;

   string = "Hello, world!";
   color = 0;     //Черный.
   horizcenter = TRUE;
   vertcenter = TRUE;

  return TRUE;
}
```

Если попытаться сейчас оттранслировать программу и запустить ее на выполнение, то пользователь не сможет изменить значения переменных, сделанные программой по умолчанию. Чтобы предоставить ему эту возможность (ведь ему может и надоесть когда-нибудь этот пуританский стиль), нужно отредактировать текст функции, работающей с диалоговым окном.

Модификация метода OnToolsOptions()

Метод OnToolsOptions() устанавливает переменные-члены класса диалогового окна соответственно значениям в документе и затем отображает все параметры на экране вместе с самим диалоговым окном. Если пользователь завершит работу с диалоговым окном, щелкнув на OK, полученные от него данные будут переписаны в переменные-члены объекта класса документа, а изображение в окне документа будет перерисовано. В текст функции OnToolsOptions() нужно добавить три оператора перед тем, как выводится на экран диалоговое окно, и три оператора — в блок, который отрабатывается после получения сообщения о шелчке на OK. Новый вариант текста OnToolsOptions()представлен в листинге 8.13.

Листинг 8.13. Файл ShowStringDoc.CPP — метод OnToolsOptions()

```
void CShowStringDoc::OnToolsOptions()
{
   COptionsDialog dlg;
   dlg.m_string = string;
   dlg.m_color = color;
   dlg.m_horizcenter = horizcenter;
   dlg.m_vertcenter = vertcenter;

   if (dlg.DoModal() == IDOK)
   {
      string = dlg.m_string;
      color = dlg.m_color;
      horizcenter = dlg.m_horizcenter;
      vertcenter = dlg.m_vertcenter;
      SetModifiedFlag();
      UpdateAllViews(NULL);
   }

}
```

А что случится, если пользователь вызовет на экран диалоговое окно и из чистого любопытства сбросит флажок Center Horizontally? Главная программа через Dialog Data Exchange (DDX), как это установлено ClassWizard, изменит значение переменной COptionsDialog::m_horizcenter на FALSE. Этот код в функции OnToolsOptions() перепишет-

ся в CShowStringDoc::horizcenter. Когда пользователь сохранит документ, метод Serialize() сохранит и CShowStringDoc::horizcenter. Все это очень увлекательно, но вот беда — на экране-то в окне документа все останется по-старому! А все потому, что мы еще не разобрались с методом OnDraw().

Модификация метода OnDraw()

Нам придется несколько усложнить эту функцию. Раньше в ней был единственный вызов функции DrawText(). Теперь нужно будет, используя параметры форматирования (новые переменные), изменить формат изображения. Чтобы приступить к редактированию OnDraw(), нужно развернуть класс CShowStringView в окне ClassView и сделать двойной щелчок на ее имени.

Цвет устанавливается функцией CDC::SetTextColor() перед обращением к DrawText(). Хорошим тоном в программировании считается сохранить прежние параметры при установке новых, а затем их восстановить. В частности, именно так мы поступим с цветом. Аргумент функции SetTextColor() имеет тип COLORREF. Можно непосредственно задать комбинацию исходных (чистых) цветов — красного, зеленого и синего — в виде шестнадцатеричного числа. Формат комбинации красного интенсивностью 0xrr, зеленого интенсивностью 0xgg и синего интенсивностью 0xbb (0x00bbggrr, т.е. 0x000000FF) — это чистый красный цвет максимальной интенсивности. Большинство программистов предпочитают использовать макрос RGB(), который принимает в качестве аргументов шестнадцатеричные числа в диапазоне 0x0–0xFF — параметры каждого из составляющих чистых цветов в порядке "красный, зеленый, синий". Так, ярко-красный цвет создается вызовом этого макроса в форме RGB(0xFF, 0, 0). Включите в текст функции OnDraw() операторы, представленные в листинге 8.14, перед вызовом функции DrawText().

Листинг 8.14. Файл ShowStringDoc.CPP — исправления в OnDraw() перед вызовом функции DrawText()

```
COLORREF oldcolor;
switch (pDoc->GetColor())
{
case 0:
   oldcolor = pDC->SetTextColor(RGB(0,0,0)); //Черный.
   break;
case 1:
   oldcolor = pDC->SetTextColor(RGB(0xFF,0,0)); //Красный.
   break;
case 2:
   oldcolor = pDC->SetTextColor(RGB(0,0xFF,0)); //Зеленый.
   break;
}
```

После вызова функции DrawText()добавьте следующий оператор:

```
pDC->SetTextColor( oldcolor);
```

Что касается центрирования текста, то существует два подхода к решению этой проблемы. Первый подход — перечислить все возможные комбинации значений двух двоичных переменных в соответствующих вариантах значений аргументов вызова функции DrawText(). Недостаток этого подхода заключается в том, что при добавлении в будущем еще каких-либо настроек придется менять весь этот фрагмент программы. Более надежный подход — установить значение специальной локальной переменной соответственно полученным параметрам настройки, объединяя их по *ИЛИ*, а затем использовать эту переменную как аргумент вызова DrawText(). Именно такой вариант показан в листинге 8.15.

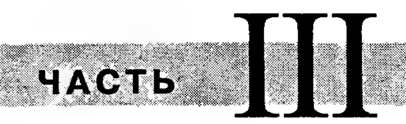
Листинг 8.15. Файл ShowStringDoc.CPP — исправления в OnDraw() после вызова функции DrawText()

```
int DTflags = 0;
if (pDoc->GetHorizcenter())
{
   DTflags |= DT_CENTER;
}
if (pDoc->GetVertcenter())
{
   DTflags |= (DT_VCENTER|DT_SINGLELINE);
}
```

Теперь при обращении к DrawText() используется локальная переменная DTflags:

```
pDC->DrawText( pDoc->GetString(), &rect, DTflags);
```

Наконец-то мы завершили этот длинный путь от переменных класса диалогового окна к переменным класса документа, а от них — к классу представления. Теперь изменение настройки элементов управления в диалоговом окне должно, в конце концов, привести к изменению картинки в окне представления документа. Остается проверить наше приложение в действии. Оттранслируйте его и попробуйте запустить. Надеюсь, никакого "генеральского эффекта"? Все работает, как задумано? Смело меняйте настройки в диалоговом окне и используйте самые замысловатые комбинации — в вашем распоряжении целых пять переключателей! Картинка должна чутко реагировать на любое ваше желание.



Расширение возможностей пользовательского интерфейса

В этой части...

ГЛАВА 9

Панели инструментов и строка состояния

В этой главе...

Создание панелей инструментов

Формирование строки состояния

Панели управления с расширенными возможностями

Создание хорошего пользовательского интерфейса — это едва ли не половина успеха в разработке приложения для Windows. К счастью, Visual C++ и его мастера предоставляют разработчику уникальные возможности для создания приложений, поддерживающих все привычные элементы пользовательского интерфейса, включая меню, диалоговые окна, панели инструментов и строку состояния. Тема меню и диалоговых окон уже рассматривалась в этой книге в главах 2 и 8. Данная глава посвящена вопросам создания и настройки панелей инструментов и строк состояния приложений.

Создание панелей инструментов

Пиктограммы на панелях инструментов, так же, как и элементы меню, соответствуют определенным командам. Хотя создать панель инструментов в приложении можно с помощью AppWizard, вам все же потребуется написать программный код для окончательной отделки. Дело в том, что причина этого кроется в различиях, существующих между приложениями, поскольку с помощью AppWizard можно создать только стандартную панель инструментов, в которую будут включены пиктограммы, встречающиеся в большинстве приложений. При разработке собственной панели инструментов для отражения специфики набора команд конкретного приложения вам, вероятно, потребуется вставить новые или удалить существующие пиктограммы.

Если с помощью AppWizard создать стандартное приложение, имеющее панель инструментов, то последняя будет выглядеть так, как показано на рис. 9.1. Эта панель инструментов содержит пиктограммы для наиболее распространенных команд меню File и Edit, а также пиктограмму для отображения окна About. Но как быть, если ваше приложение не поддерживает эти команды? В данном случае потребуется так модифицировать создаваемую по умолчанию панель инструментов, чтобы она соответствовала командам именно вашего приложения.

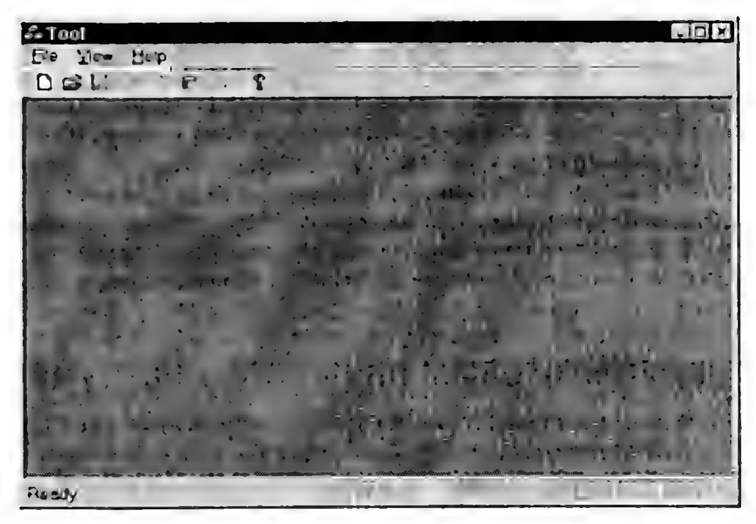


Рис. 9.1. Создаваемая по умолчанию панель инструментов содержит пиктограммы для выполнения наиболее распространенных команд

Удаление пиктограмм с панели инструментов

Создайте приложение с многооконным интерфейсом, включающее панель инструментов. Для этого выберите команду File⇨New, в открывшемся диалоговом окне выберите вкладку Projects, присвойте приложению имя Tool и примите значения по умолчанию в каждом из диалоговых окон, которые AppWizard будет выводить на экран. Для ускорения процесса можно на первом же шаге щелкнуть на кнопке Finish. Мастер AppWizard по умолчанию формирует стационарную панель инструментов. Завершите создание приложения и запустите его. На экране вы увидите панель инструментов, в точности повторяющую ту, которая изображена на рис. 9.1.

Прежде чем продолжить работу, поэкспериментируйте с этой панелью. Команда меню View позволяет убрать панель с экрана и вновь вывести ее. Выполните это. А теперь разместите указатель мыши между пиктограммами и перетащите панель куда-либо вниз в окне приложения. Отпустите кнопку мыши, и панель примет вид плавающей панели. Попробуйте переместить ее в различные позиции, а затем зафиксируйте ее возле нижней или боковой границы окна приложения. Панель станет стационарной в выбранной вами позиции. Обратите внимание, что при перетаскивании пунктирный прямоугольник изменяет свою форму, когда панель превращается из плавающей в стационарную. Вновь сделайте панель плавающей, а затем уберите ее с экрана, щелкнув на кнопке со знаком ×, расположенной в верхнем правом углу панели. Затем верните панель на экран с помощью команды меню View. Обратите внимание, что панель появилась на экране в том же месте, где она находилась в момент удаления с экрана. Все эти возможности автоматически обеспечиваются в приложениях средствами MFC и AppWizard.

В качестве первого шага в процедуре настройки панелей инструментов выполним удаление пиктограмм, которые в приложении не потребуются. Начните с того, что для отображения ресурсов приложения щелкните на корешке вкладки ResourceView. Далее щелкните на значке + рядом с элементом Toolbar, а затем сделайте двойной щелчок на ресурсе IDR_MAINFRAME. Откроется окно редактирования панелей инструментов, показанное на рис. 9.2.

После того как окно редактора панелей инструментов будет открыто, удаление пиктограмм с панели инструментов сводится к простому перетаскиванию их с панели на свободное место в окне. Поместите указатель мыши на удаляемую с панели пиктограмму, нажмите левую кнопку мыши и, удерживая кнопку нажатой, уберите пиктограмму с панели инструментов. Когда вы отпустите кнопку мыши, удаляемая пиктограмма исчезнет. В нашем приложении Tool удалите все пиктограммы, кроме Help, которая имеет вид желтого знака вопроса. На рис. 9.3 показана отредактированная панель инструментов, на которой осталась только пиктограмма Help. Еще одна пустая пиктограмма является всего лишь заглушкой, которую можно использовать как отправную точку в процедуре создания новой, определяемой вами пиктограммы. Если вы не будете с ней ничего делать, то в окончательном варианте созданной панели инструментов она исчезнет.

Добавление пиктограмм на панель инструментов

Процедура включения новой пиктограммы в панель инструментов состоит из двух этапов. На первом из них следует нарисовать изображение пиктограммы, а на втором вы должны будете связать команду с новой пиктограммой. Приступая к созданию изображения новой пиктограммы, прежде всего щелкните на заглушке пустой пиктограммы, расположенной на формируемой панели инструментов. Изображение пустой пиктограммы в увеличенном масштабе появится в окне редактирования (рис. 9.4).

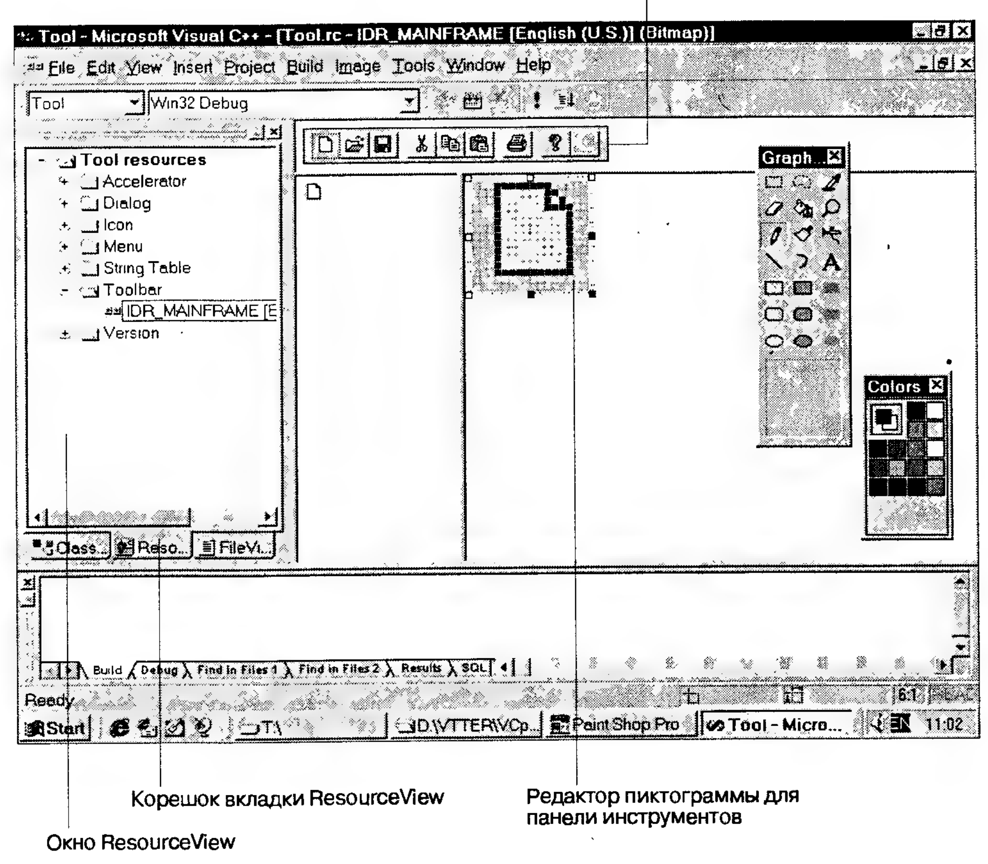


Рис. 9.2. Для настройки панелей инструментов приложения используйте редактор панелей инструментов

Предположим, что на панели инструментов требуется создать пиктограмму, которая вызывает программу вычерчивания в окне приложения красного круга. В качестве значка этой пиктограммы в окне редактора пиктограммы нарисуем красный круг (с помощью инструмента Ellipse). Выведите на экран окно свойств Properties и присвойте пиктограмме соответствующий идентификатор команды, например ID_CIRCLE.

Далее следует определить для пиктограммы содержание контекстного окна указателя (ToolTip) и ввести ее описание. Контекстное окно указателя будет выводиться на экран в том случае, когда пользователь поместит указатель мыши на данную пиктограмму и задержит его на ней на одну-две секунды. Эта процедура напомнит пользователю назначение данной пиктограммы. Для нашего примера подходящим значением для указателя будет слово Circle (Круг). Описание же пиктограммы выводится в строке состояния приложения. В нашем случае подходящим описанием будет фраза Draws a red circle in the window (Рисует в окне приложения красный круг). Введите обе эти текстовые строки в поле Prompt. Сначала вводится текст описания пиктограммы, затем — знак перехода на новую строку (\n) и текст в контекстном окне указателя для данной пиктограммы (рис. 9.5).

Теперь необходимо определить идентификатор команды, связанной с новой пиктограммой. Как правило, это идентификатор команды, уже определенной в качестве пункта меню и связанной с конкретной подпрограммой. В этом случае все, что от вас требуется, — выбрать

существующий идентификатор команды из раскрывающегося списка. Описание будет взято из свойств выбранного пункта меню, а обработчик сообщения для пункта меню уже был определен ранее. В нашем примере приложения пиктограмма, установленная на панель инструментов, не соответствует какому-либо существующему пункту меню. Поэтому нужно связать идентификатор с функцией обработки сообщения, которую MFC будет автоматически вызывать после того, как пользователь сделает щелчок на пиктограмме.

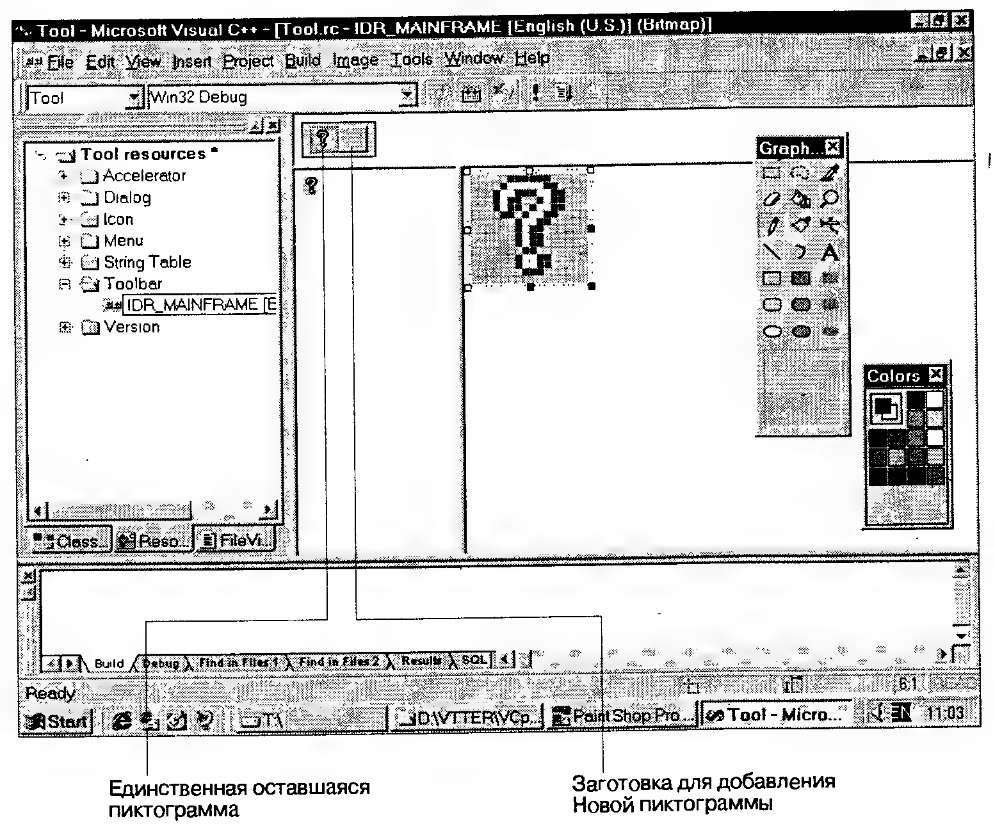


Рис. 9.3. На редактируемой панели инструментов осталась только одна пиктограмма (пустая заглушка для новой пиктограммы в расчет не принимается)

Для этого выполните следующие действия.

1. Убедитесь, что пиктограмма, для которой вы хотите создать программу обработки сообщения, выбрана на пользовательской панели инструментов, а затем вызовите ClassWizard.
2. На экран будет выведена вкладка свойств ClassWizard, на которой уже выбран идентификатор данной пиктограммы (рис. 9.6). Для добавления функции обработки сообщения выберите в списке Class name класс, к которому вы хотите добавить эту функцию в качестве члена (в нашем примере используется класс представления).
3. В списке Messages дважды щелкните на элементе COMMAND.
4. Примите имя функции, которое MFC предложит по умолчанию в следующем окне сообщения, и все необходимые определения будут сделаны. Для фиксации изменений щелкните на кнопке OK.

На заметку

Если вы не определите функцию обработки сообщения, связанную с пиктограммой на панели инструментов, или же если не будет существовать ни одного экземпляра класса, который перехватывает это сообщение, MFC при запуске приложения отобразит данную пиктограмму заблокированной (т.е. ее изображение будет затенено). Например, если в приложении с многооконным интерфейсом сообщение перехватывается документом или представлением и ни один из документов не открыт, пиктограмма будет неактивной. Это же справедливо и для команд меню — пиктограммы панелей инструментов при всех обстоятельствах фактически являются эквивалентами команд меню.

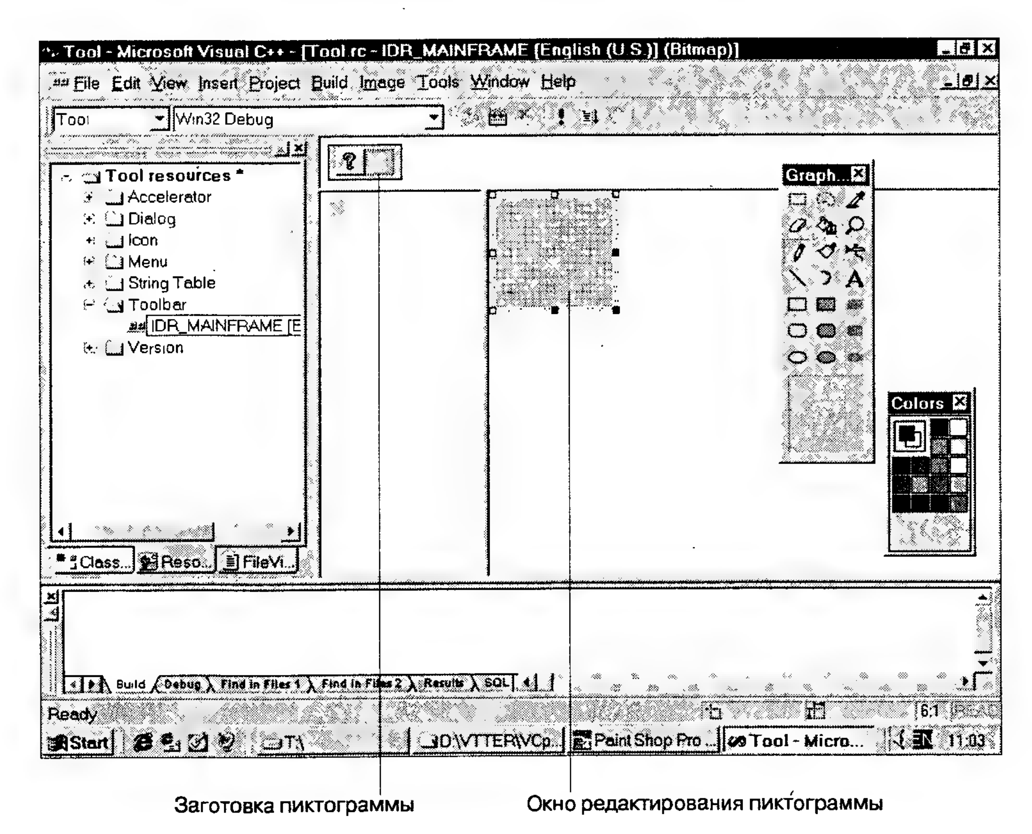


Рис. 9.4. Щелкнув на заглушке пустой пиктограммы, можно раскрыть окно редактора пиктограммы

На заметку

Как правило, пиктограммы панелей инструментов дублируют команды меню, обеспечивая пользователю более быстрый способ вызова чаще всего используемых команд меню. В такой ситуации элемент меню и пиктограмма панели инструментов представляют одну и ту же команду и вы присваиваете им одинаковые идентификаторы. В этом случае, как при выборе пользователем команды меню, так и после щелчка на пиктограмме панели инструментов вызывается одна и та же функция обработки сообщения.

Если теперь откомпилировать и запустить приложение, то на экране можно будет увидеть окно, показанное на рис. 9.7. На этом рисунке на панели инструментов уже присутствует новая пиктограмма, для которой в контекстном окне указателя отображено название, а в строке состояния выведено описание. Панель инструментов в данном случае выглядит несколько бедновато, но у вас всегда есть возможность поместить на нее новые пиктограммы.

Рис. 9.5. После подготовки изображения пиктограммы следует определить ее свойства

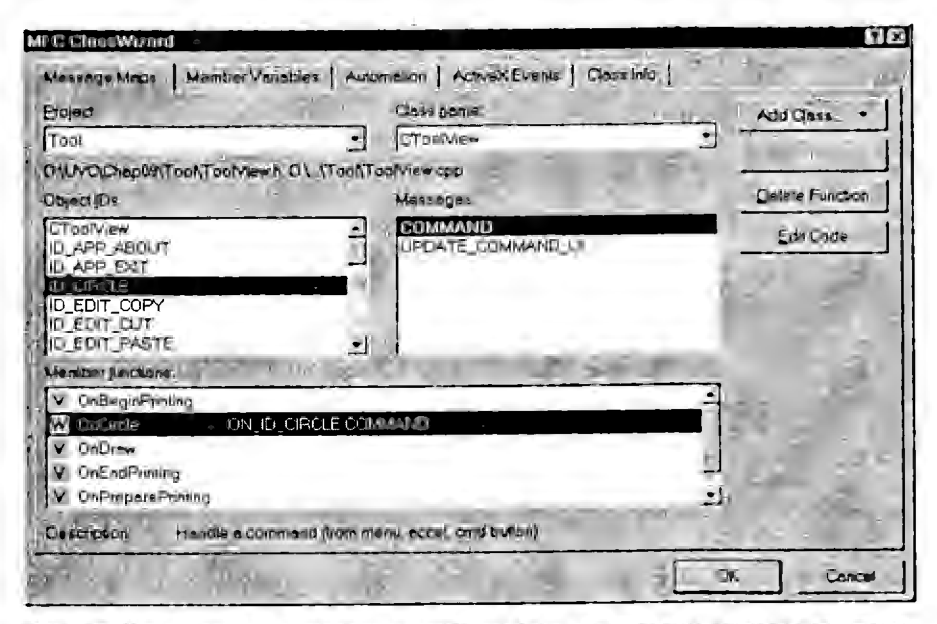


Рис. 9.6. Для организации перехвата сообщений от пиктограмм панелей инструментов можно использовать `ClassWizard`

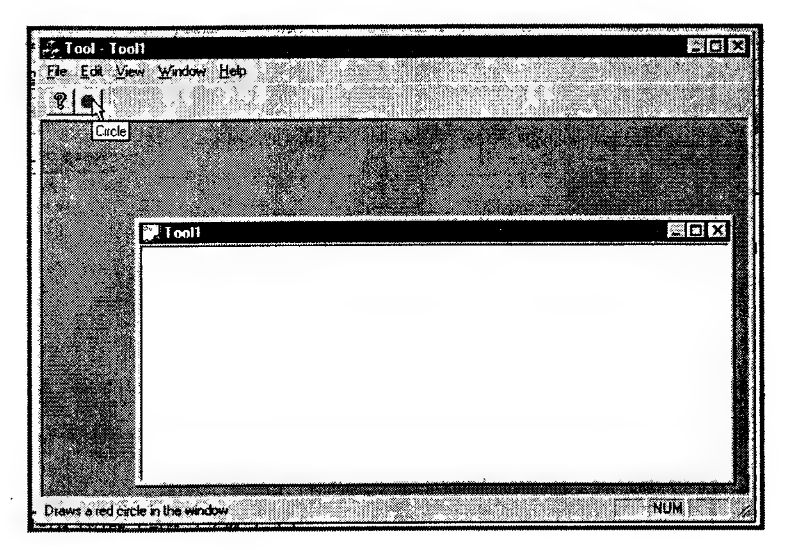


Рис. 9.7. Новая панель инструментов с отображенными на ней названием и описанием пиктограммы

Следуя описанной выше процедуре, можно создать на панели инструментов столько новых пиктограмм, сколько посчитаете нужным. После того как пиктограммы будут созданы, необходимо обратиться к ресурсам панели инструментов, предварительно подготовив тексты подпрограмм, которые будут вызываться для обработки щелчков на соответствующих пиктограммах. Например, в случае нашей пиктограммы `Circle`, установленной на панель инструментов, к программе следует добавить функцию обработки сообщения с именем `OnCircle()`. MFC будет вызывать эту функцию всякий раз, когда пользователь сделает щелчок на соответствующей пиктограмме панели инструментов. Однако на данной стадии разработки эта функция не выполняет абсолютно ничего (листинг 9.1).

Листинг 9.1. Пустая функция обработки сообщения

```
void CToolView::OnCircle()
{
  // TODO: поместите сюда текст подпрограммы обработки события.
}
```

Поскольку предполагается, что пиктограмма Circle будет использована для обрисовки красного круга в окне приложения, следует, очевидно, позаботиться о подготовке функции `OnCircle()`, которая могла бы выполнить это задание. Поместите в текст данной функции операторы, приведенные в листинге 9.2, после чего пиктограмма в работающем приложении сможет реально выполнять то, что от нее ожидается (результат показан на рис. 9.8).

Листинг 9.2. Функция `CToolView::OnCircle()`

```
void CToolView::CToolView()
{
  CClientDC clientDC(this);
  Cbrush newBrush(RGB(255,0,0));
  CBruh* oldBrush = clientDC.SelectObject(&newBrush);
  clientDC.Ellipse(20, 20, 200, 200);
  clientDC.SelectObject(oldBrush);
}
```

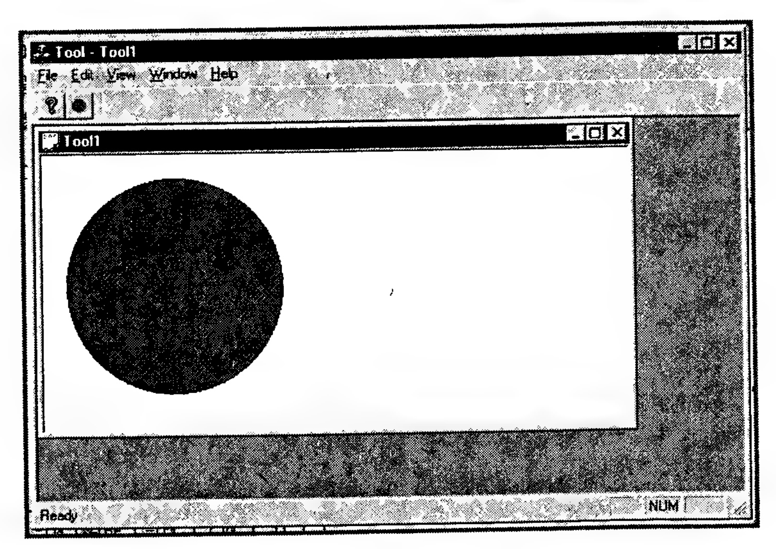


Рис. 9.8. После ввода текста функции `OnCircle()` *приложение отреагирует на щелчок на новой пиктограмме*

Методы класса CToolBar

В большинстве случаев после того, как вы определите ресурсы панели инструментов и свяжете ее пиктограммы с идентификаторами соответствующих команд, ваша работа с панелью инструментов будет закончена. Тексты программ, подготовленные для вас AppWizard, обеспечат создание панели инструментов, а MFC при необходимости позаботится о вызове соответствующих функций обработки сообщений. Тем не менее может возникнуть ситуация, при которой вам потребуется некоторым образом изменить принятый по умолчанию порядок работы панели инструментов. В этом случае у вас есть возможность обратиться к функциям, входящим в состав класса CToolBar, которые вместе с кратким описанием перечислены в табл. 9.1. Панель инструментов доступна из класса CMainFrame как переменная-член m_wndToolBar. Как правило, порядок работы панели инструментов корректируется функцией CMainFraim::OnCreate().

Таблица 9.1. Функции-члены класса CToolBar

Функция	Назначение
CommandToIndex()	Возвращает индекс пиктограммы соответственно ее идентификатору
Create()	Создает панель инструментов
GetButtonInfo()	Возвращает информацию о пиктограмме
GetButtonStyle()	Возвращает стиль пиктограммы
GetButtonText()	Возвращает текст названия пиктограммы
GetItemID()	Возвращает идентификатор пиктограммы соответственно ее индексу в панели
GetItemRect()	Возвращает параметры прямоугольника, ограничивающего элемент, индекс которого задается как аргумент функции
GetToolBarCtrl()	Возвращает ссылку на объект класса CToolBarCtrl, представленный объектом CToolBar

Функция	Назначение
LoadBitmap()	Загружает графические изображения пиктограмм
LoadToolBar()	Загружает ресурсы панели инструментов
SetBitmap()	Задает графические образы пиктограмм новой панели инструментов
SetButtonInfo()	Определяет идентификатор, стиль и номер графического образа пиктограммы
SetButtons()	Задает идентификаторы для пиктограмм панели инструментов
SetButtonStyle()	Задает стиль пиктограммы
SetButtonText()	Задает текст названия пиктограммы
SetHeight()	Задает высоту панели инструментов
SetSizes()	Задает размеры пиктограмм

Как правило, у вас не возникнет необходимости вызывать методы панели инструментов, но, обращаясь к ним, можно достичь некоторых необычных результатов, например увеличить размеры панели инструментов, как показано на рис. 9.9. (Размер пиктограмм сохранен, но сама панель инструментов стала больше.) Подобный вид панели инструментов получен путем обращения к методу SetHeight() объекта — панели инструментов. Методы класса CToolBar дают возможность выполнять с панелями инструментов различные трюки, но пользоваться ими следует с большой осторожностью.

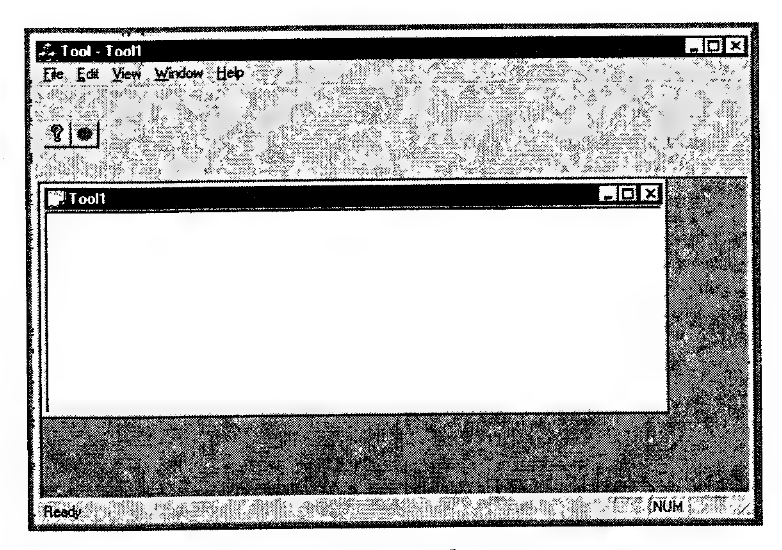


Рис. 9.9. Функции, входящие в состав объекта панели инструментов, используются для изменения внешнего вида и функционирования панели инструментов

Формирование строки состояния

Строки состояния являются очень полезными объектами, которые располагаются в нижней части окон приложений. Функция строки состояния (как она определена MFC) — отображение описания команд и состояния различных клавиш клавиатуры, включая клавиши

<Caps Lock> и <Scroll Lock>. Фактически строка состояния настолько стандартна с точки зрения программистов (это справедливо, по крайней мере, для приложений, создаваемых с помощью AppWizard), что для нее даже отсутствуют ресурсы, которые можно было бы редактировать, как в случае с панелью инструментов. Достаточно только потребовать от AppWizard включить строку состояния в приложение, и вся необходимая работа будет автоматически выполнена.

Но правильно ли это? Строка состояния, так же, как и панели инструментов, должна отражать специфические особенности вашего приложения. Для подобных целей в класс CStatusBar включен набор методов, которые дают возможность настраивать внешний вид строки состояния и выполняемые ею функции. В табл. 9.2 содержится перечень этих методов и их краткое назначение.

Таблица 9.2. Методы класса CStatusBar

Метод	Назначение
CommandToIndex()	Возвращает индекс индикатора, заданного его идентификатором
Create()	Создает строку состояния
GetItemID()	Возвращает идентификатор индикатора, заданного его индексом
GetItemRect()	Возвращает параметры прямоугольника элемента, заданного его индексом
GetPaneInfo()	Возвращает информацию об индикаторе
GetPaneStyle()	Возвращает стиль индикатора
GetPaneText()	Возвращает текст индикатора
GetStatusBarCtrl()	Возвращает ссылку на объект класса CStatusBarCtrl, представленный объектом CStatusBar
SetIndicators()	Задает идентификатор индикатора
SetPaneInfo()	Задает для индикатора идентификатор, ширину и стиль
SetPaneStyle()	Задает стиль индикатора
SetPaneText()	Задает текст индикатора

Если при настройке AppWizard задать включение в приложение строки состояния, будет создано окно приложения, подобное изображенному на рис. 9.10. (Для самостоятельной подготовки такого рода приложения создайте проект с именем Status и оставьте все настройки по умолчанию, как вы уже делали в отношении приложения Tool.) Строка состояния разделена на несколько частей, называемых ячейками, которые отображают определенную информацию о состоянии приложения и системы. Эти ячейки, отмеченные на рис. 9.10, включают индикаторы для клавиш <Caps Lock>, <Num Lock> и <Scroll Lock>, а также область сообщений, в которой отображаются данные о состоянии и описание команд. Чтобы увидеть описание команды, следует поместить указатель мыши на пиктограмму панели инструментов (рис. 9.11).

Наиболее распространенный метод настройки строки состояния — это добавление в нее новых ячеек. Для добавления ячейки в строку состояния следует выполнить следующее.

1. Создать идентификатор команды для новой ячейки.
2. Подготовить текст, помещаемый в ячейку по умолчанию.
3. Добавить идентификатор команды ячейки в массив индикаторов строки состояния.
4. Создать для ячейки функцию обработки команды обновления.

В последующих разделах каждая из упомянутых операций рассматривается детально.

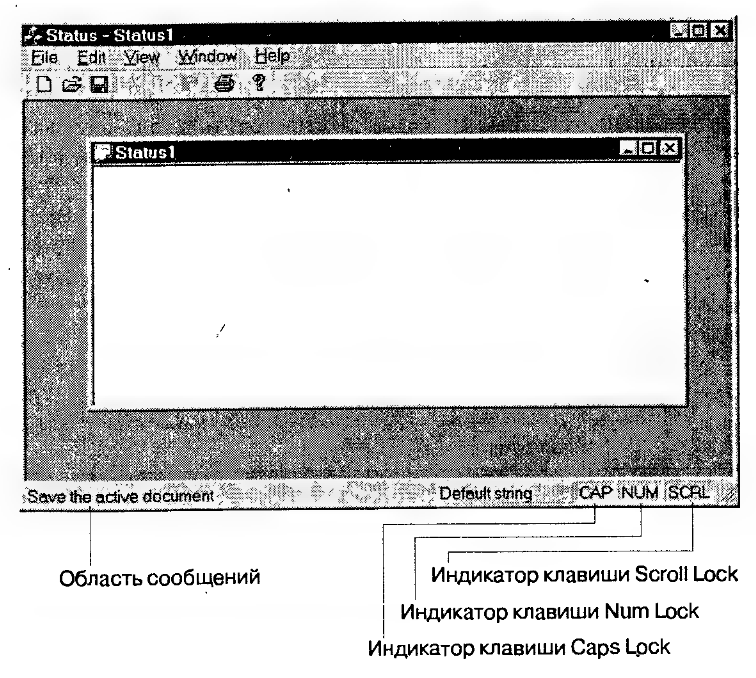


Рис. 9.10. Создаваемая в MFC по умолчанию строка состояния содержит несколько информационных ячеек

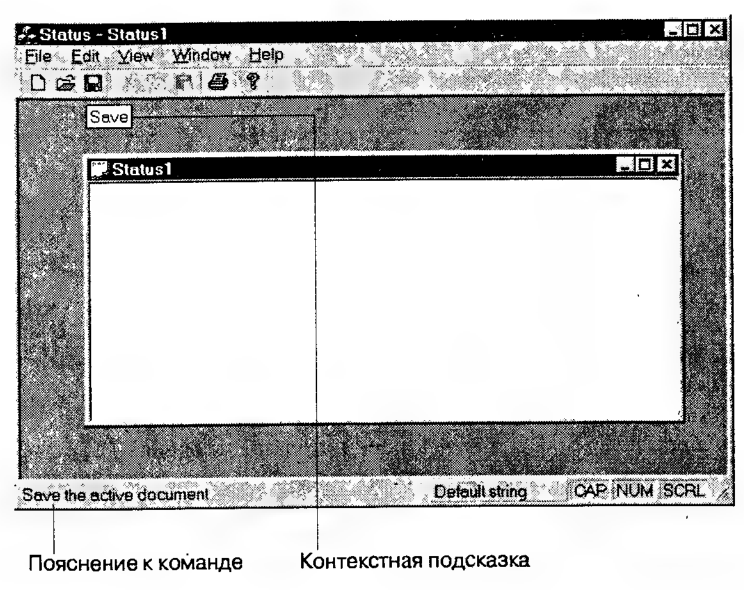


Рис. 9.11. Область сообщений чаще всего используется для отображения описания команд

Создание нового идентификатора команды

Эта операция не представляет большой сложности благодаря наличию в Visual C++ средства для просмотра (броузера) символов. Добавление идентификатора новой команды начните с выбора команды View⇨Resource Symbols (Просмотр⇨Символы ресурсов). Откроется диалоговое окно Resource Symbols (рис. 9.12), в котором будут отображены символы, определенные на данный момент для ресурсов приложения. Для того чтобы открыть диалоговое окно New Symbol (Новый символ), щелкните на кнопке New. В поле Name введите новый идентификатор, например для нашего случая — ID_MYNEWPANE (рис. 9.13). Как правило, в качестве идентификатора достаточно просто принять то значение, которое предлагает MFC.

Для завершения работы щелкните на кнопке OK окна New Symbol, а затем на кнопке Close окна Resource Symbols, и новый идентификатор команды будет определен.

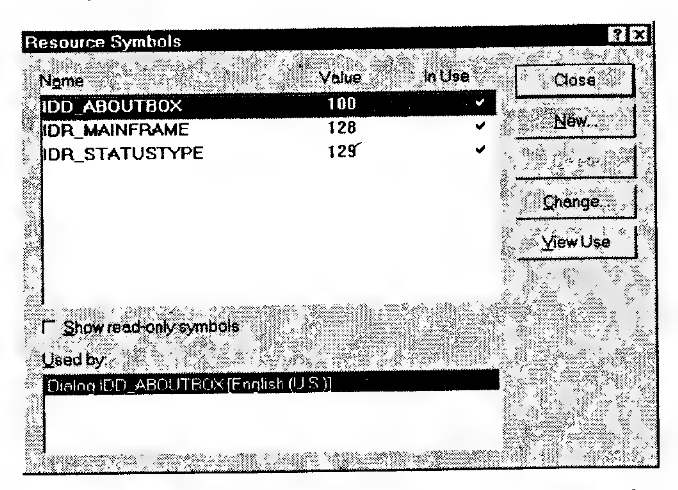


Рис. 9.12. Для добавления в приложение нового идентификатора команды воспользуйтесь диалоговым окном Resource Symbols

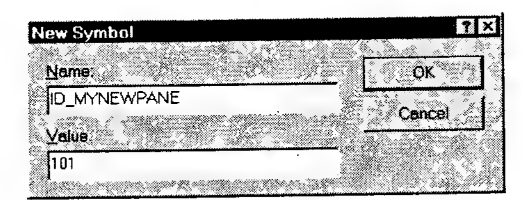


Рис. 9.13. В диалоговом окне New Symbol *введите имя и значение нового идентификатора*

Определение текста, отображаемого по умолчанию

Вы уже определили идентификатор ресурса, но он еще нигде не используется. Для идентификатора, определяющего ячейку строки состояния, обязательно должно быть задано значение, выводимое в нее по умолчанию. Чтобы задать этот текст, следует открыть окно ResourceView (щелкнув на корешке вкладки ResourceView в левой части окна среды разработки) и сделать двойной щелчок на ресурсе String Table. На экране откроется окно редактора таблицы строк, показанное на рис. 9.14.

Далее, для того чтобы открыть диалоговое окно String Properties, выберите команду Insert⇨New String (Вставка⇨Новая строка). В поле ID (рис. 9.15) ведите идентификатор команды новой ячейки (или выберите его из раскрывающегося списка), а в поле Caption введите само значение по умолчанию (в нашем случае — Default string).

Добавление идентификатора в массив индикаторов

MFC использует массив идентификаторов в процессе формирования строки состояния приложения для определения того, где и какие ячейки должны быть отображены. Этот массив идентификаторов передается в качестве аргумента методу класса строки состояния Set_Indicators(), который, в свою очередь, вызывается методом OnCreate() класса CMainFrame. Вы можете найти этот массив, показанный в листинге 9.3, в начале файла MainFrm.cpp. Вот один из способов получить доступ к этим строкам в окне редактора программ: переключитесь на вкладку ClassView, раскройте класс CMainFraim, дважды щелкните на функции OnCreate() и перейдите на одну страницу вверх (к началу файла). Альтернативный вариант — воспользуйтесь вкладкой FileView и откройте файл MainFrm.cpp.

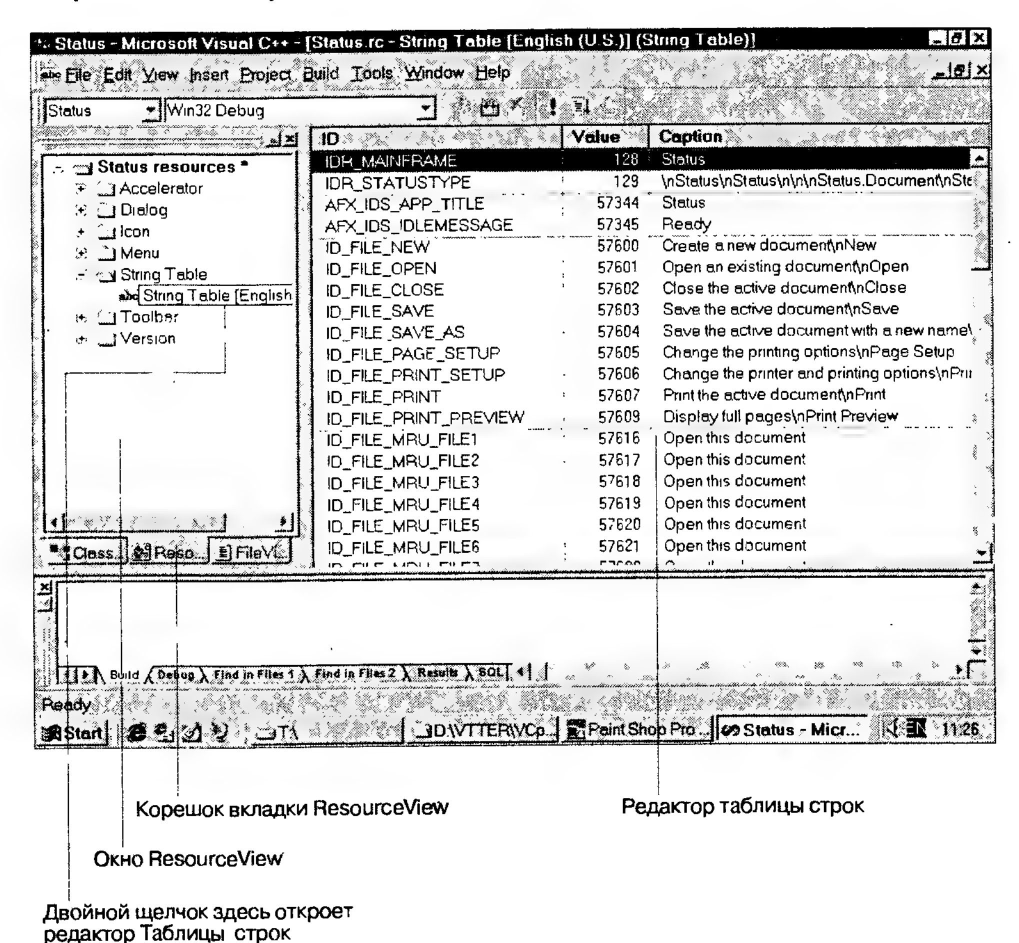


Рис. 9.14. В таблице строк задайте строку, которая по умолчанию будет выводиться в новую ячейку строки состояния

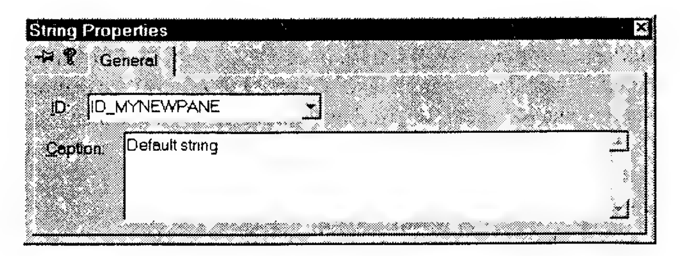


Рис. 9.15. Для определения значения, выводимого в новую ячейку по умолчанию, используйте диалоговое окно String Properties

Листинг 9.3. Файл MainFrm.cpp — массив индикаторов

```
static UINT indicators[] =
{
   ID_SEPARATOR, // Индикатор строки состояния.
   ID_INDICATOR_CAPS,
   ID_INDICATOR_NUM,
   ID_INDICATOR_SCRL,
};
```

Для добавления к массиву новой ячейки введите в него ее идентификатор, причем его положение в массиве должно соответствовать позиции ячейки в строке состояния. (Первая ячейка, ID_SEPARATOR, всегда должна оставаться на первой позиции.) В листинге 9.4 приведен пример массива индикаторов, в который добавлена новая ячейка.

Листинг 9.4. Файл MainFrm.cpp — расширенный массив индикаторов

```
static UINT indicators[] =
{
   ID_SEPARATOR, // Индикатор строки состояния.
   ID_MYNEWPANE,
   ID_INDICATOR_CAPS,
   ID_INDICATOR_NUM,
   ID_INDICATOR_SCRL,
};
```

Создание функции обработки команды обновления для ячейки индикатора

При формировании строки состояния MFC не разблокирует автоматически новую ячейку. Вы сами должны создать для новой ячейки функцию обработки команды обновления и в ней активизировать эту ячейку. (Мы уже знакомились с обработчиками команд обновления в главе 3.) Кроме того, в большинстве приложений текст, выводимый в строку состояния, формируется динамически — значение по умолчанию, определенное нами на предыдущем этапе, является всего лишь заглушкой.

Обычно для организации перехвата сообщений мы пользуемся мастером ClassWizard. Однако, когда речь идет о перехвате сообщения строки состояния, ClassWizard нам не помощник. В этом случае потребуется вручную добавить необходимые элементы в карту сообщений, а затем написать саму функцию обработки. Не забудьте добавить соответствующие элементы и в карту сообщений в файле заголовка, и в файл текста программы, причем добавлять их следует вне специального комментария AFX_MSG_MAP, используемого мастером ClassWizard.

Для открытия файла заголовка сделайте двойной щелчок на имени класса CMainFrame в окне ClassView и пролистайте содержимое файла в окне редактора кода до самого конца. Отредактируйте карту сообщений так, как это показано в листинге 9.5. При создании собственных приложений для обновления ячеек строки состояния можно использовать функции с произвольными именами, но остальная часть объявления должна будет всегда иметь тот же самый вид.

Листинг 9.5. Файл MainFrm.h — карта сообщений

```
// Генерация функций карты сообщений.
protected:
  //{{AFX_MSG(CMainFrame)
  afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    // ВНИМАНИЕ! Функции здесь добавляет и удаляет ClassWizard.
    // НЕ РЕДАКТИРУЙТЕ то, что находится


    // в этом блоке сгенерированного кода!
  //}}AFX_MSG
  afx_msg void OnUpdateMyNewPane(CCmdUI *pCmdUI);
  DECLARE_MESSAGE_MAP()
```

Далее для действительного связывания идентификатора команды с обработчиком следует добавить обработчик к карте сообщений в файле реализации. Откройте текст любой из функций класса CMainFrame и перейдите в самое начала файла. Где-то там обычно и размещается карта сообщений. Отредактируйте карту так, как это показано в листинге 9.6.

Листинг 9.6. Файл MainFrm.cpp — карта сообщений

```
BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
  //{{AFX_MSG_MAP(CMainFrame)
    // ВНИМАНИЕ! Макросы назначений здесь добавляет и
    // удаляет ClassWizard. НЕ РЕДАКТИРУЙТЕ то, что
    // находится в этом блоке сгенерированного кода!
  ON_WM_CREATE()
  //}}AFX_MSG_MAP
  ON_UPDATE_COMMAND_UI(ID_MYNEWPANE, OnUpdateMyNewPane)
END_MESSAGE_MAP()
```

Теперь обработка организована так, что, когда потребуется обновить ячейку строки состояния ID_MYNEWPANE, будет вызвана метод OnUpdateMyNewPane() объекта класса CMainFrame.

На данный момент уже все готово для того, чтобы написать новый обработчик команды обновления. В этом обработчике вы будете активизировать новую ячейку и определять ее содержание. В листинге 9.7 приведен пример обработчика команды обновления для новой ячейки строки состояния. Видно, что он использует переменную-член m_paneString. Подобные обработчики должны работать очень быстро — задача контроля корректности строки, помещаемой в переменную m_paneString, должна быть возложена на другую функцию, которая будет вызываться намного реже.



Функции обработки команд обновления подробно рассматриваются в главе 3. Эти функции должны работать очень быстро, поскольку они вызываются при любом обновлении изображения на экране.

Листинг 9.7. Функция CMainFrame::OnUpdateMyNewPane()

```
void CMainFrame::OnUpdateMyNewPane(CCmdUI *pCmdUI)
{
  pCmdUI->Enable();
  pCmdUI->SetText(m_paneString);
}
```

Определение внешнего вида строки состояния

Создавая в нашем приложении строку состояния, в качестве последнего штриха определим способ задания значения переменной m_paneString. Для ее инициализации добавьте в конструктор CMainFrame следующую строку:

```
m_paneString = "Default string";
```

Значение, которое было введено в таблицу строк, предназначено всего лишь для того, чтобы среде разработки было известно, что созданный идентификатор ресурса будет использоваться. Для добавления закрытой (private) переменной-члена m_paneString щелкните в окне ClassView правой кнопкой мыши на классе CMainFrame и выберите команду Add Member Variable. Для определяемой переменной следует указать тип CString.

Для инициализации строки состояния поместите в функцию CMainFrame::OnCreate() непосредственно перед оператором return приведенные ниже операторы:

```
CClientDC dc(this);
SIZE size = dc.GetTextExtent(m_paneString);
int index = m_wndStatusBar.CommandToIndex(ID_MYNEWPANE);
m_wndStatusBar.SetPaneInfo(index, ID_MYNEWPANE,
    SBPS_POPOUT, size.cx);
```

Эти операторы определяют текст и размер ячейки строки состояния. Размер ячейки задается при вызове функции SetPaneInfo(), которой передается индекс ячейки и ее новый размер. Индекс ячейки можно получить с помощью функции CommandToIndex(), а функция GetTextExtent() предоставит вам ее размер. Как элемент отделки при вызове SetPaneInfo() можно использовать стиль SBPS_POPOUT. В результате данная ячейка будет отображена в строке состояния выпуклой, а не просто с отступом.

Значение в строке будет изменяться, когда пользователь выберет соответствующую команду меню. В окне редактора ресурсов откройте меню IDR_STATUSTYPE и добавьте элемент Change String в меню File. (Работа с меню уже рассматривалась нами ранее, в главе 8.) Назначьте ему идентификатор ресурса ID_FILE_CHANGESTRING.

Перейдите в окно ClassWizard и добавьте функцию обработки для этой команды. Она должна перехватываться объектом класса CMainFrame, поскольку именно в нем содержится переменная m_paneString. В окне ClassWizard для этой функции вам будет предложено имя OnFileChangestring(), которое следует принять. Щелкните на кнопке OK, чтобы окно ClassWizard закрылось.

Добавьте в приложение новое диалоговое окно и присвойте ему идентификатор IDD_PANEDLG. В качестве его заголовка введите текст Изменение значения ячейки строки состояния. Поместите в это окно единственное текстовое поле, растянутое по всей ширине диалогового окна, и присвойте ему идентификатор IDC_EDIT1. Разместите над текстовым полем статический текст, содержащий заголовок **Новая строка**. Оставив в окне редактора ресурсов создаваемое диалоговое окно открытым, вызовите ClassWizard. Создайте для диалогового окна новый класс с именем CPaneDlg и свяжите поле IDC_EDIT1 с переменной-членом класса диалога m_paneString и типа CString.



Добавление к приложению диалоговых окон и связывание с ними классов подробно обсуждалось в главах 2 и 8.

Перейдите на вкладку ClassView и раскройте класс CMainFrame, а затем сделайте двойной щелчок на методе OnFileChangeString(), чтобы начать его редактирование. Добавьте в него операторы, приведенные в листинге 9.8.

Листинг 9.8. Функция CMainFrame::OnFileChangestring()

```
void CMainFrame::OnFileChangestring()
{
  CPaneDlg dialog(this);
  dialog.m_paneString = m_paneString;

  int result = dialog.DoModal();

  if (result == IDOK)
  {
    m_paneString = dialog.m_paneString;
    CClientDC dc(this);
    SIZE size = dc.GetTextExtent(m_paneString);
    int index = m_wndStatusBar.CommandToIndex(ID_MYNEWPANE);
    m_wndStatusBar.SetPaneInfo(index,
      ID_MYNEWPANE, SBPS_POPOUT, size.cx);
  }
}
```

Этот фрагмент программы выводит на экран диалоговое окно и, если пользователь завершает работу в этом окне, щелкнув на кнопке OK, меняются текст и размеры ячейки строки состояния. Данный фрагмент программы очень похож на тот, который был включен в метод OnCreate(). Вставьте в начало файла MainFrm.cpp следующую строку:

```
#include "panedlg.h"
```

В результате компилятор получит доступ к определению класса CPaneDlg. Выполните компиляцию и запустите приложение Status. На экране появится окно, изображенное на рис. 9.16. Как видите, строка состояния содержит дополнительную ячейку, в которую помещен текст Default string. Если в окне приложения Status выбрать команду File⇨ Change String, раскроется диалоговое окно, в котором можно ввести новый текст для строки состояния. Если закрыть это диалоговое окно, щелкнув на кнопке OK, введенный текст появится в строке состояния, а размер предназначенной для него ячейки изменится в соответствии с длиной этого текста (рис. 9.17).

Панели управления с расширенными возможностями

В последнее время при создании приложений все чаще используется новый дизайн панелей управления, которые содержат не только пиктограммы, но и элементы управления других типов. Такие панели получили в оригинальной документации наименование ***Rebar***. В уже привычную нам панель инструментов тоже можно добавить элементы управления, отличные от пиктографических кнопок, но сделать это очень непросто. А вот включение в rebar разнообразных элементов управления выполняется довольно просто.

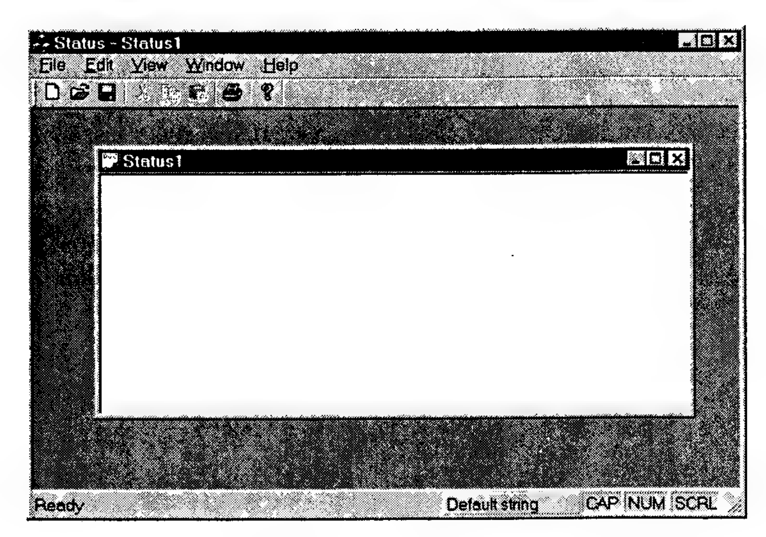


Рис. 9.16. Окно приложения, демонстрирующее методы добавления в строку состояния новых ячеек и работу с ними

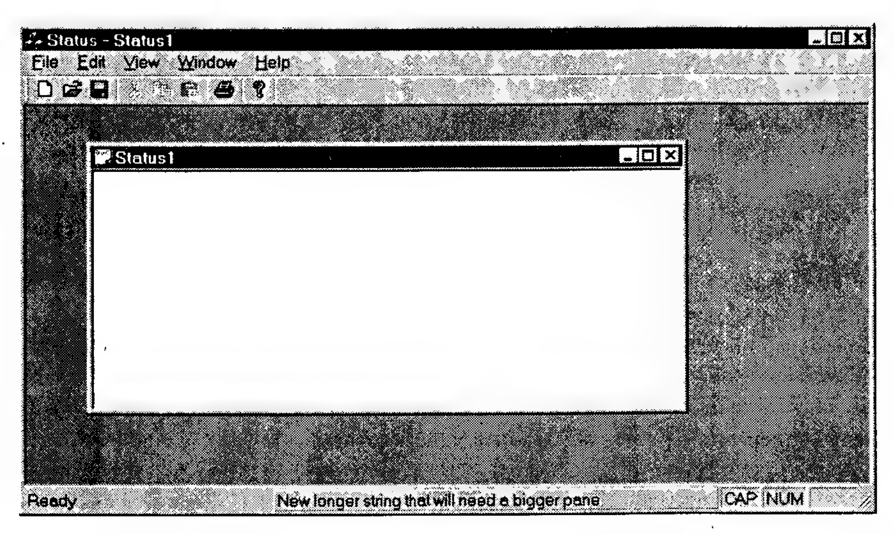


Рис. 9.17. Размеры ячейки строки состояния меняются в соответствии с длиной помещенного в нее текста

При помощи AppWizard создайте новый проект ReBar. Как и ранее, примите без возражений все настройки, которые мастер предлагает по умолчанию, или щелкните на Finish в окне первого этапа настройки. Когда заготовка проекта будет сформирована, дважды щелкните на имени класса CMainFrame в окне ClassView и отредактируйте файл заголовка. Сформированный класс фрейма имеет классическую панель инструментов. В него же мы включим и новый rebar. Для начала включим в определение класса новый открытый (public) член-переменную:

```
CReBar m_rebar;
```

В этом простом примере мы включим в панель простейший элемент управления — флажок. Но в своих экспериментах вы можете по той же методике включить и любой другой элемент. И флажок, и переключатель, и командная кнопка (наподобие привычной для всех OK) представляют собой экземпляры класса CButton и различаются только стилем. Добавьте в файл заголовка член-переменную для флажка сразу же после переменной для панели:

```
CButton m_check;
```

В предыдущем разделе уже было сказано, что панель инструментов в приложении создается и инициализируется классом фрейма главного окна. Это же справедливо и в отношении rebar. Добавьте в метод OnCreate() класса CMainFrame перед оператором return следующие строки кода:

```
if( !m_rebar.Create(this))
{
  TRACE0("Failed to create rebar\n")
  return -1;      //Панель не сформирована.
}
```

Для флажка необходимо сформировать идентификатор ресурса. При включении элементов управления в диалоговое окно при помощи редактора диалога имя, присвоенное элементу, автоматически связывалось с числом. В данном же случае мы не можем воспользоваться средствами автоматической генерации идентификаторов и должны вручную определить все необходимые значения непосредственно в тексте программы. Точно так мы поступили в предыдущем разделе с идентификатором индикатора строки состояния. Выберите в меню среды разработки View⇨Resource Symbols и щелкните на кнопке New. Введите имя IDC_CHECK и примите предлагаемое числовое значение идентификатора. Затем добавьте в файл resource.h строку, определяющую значение IDC_CHECK. Теперь можно быть уверенным, что другие элементы управления в приложении не получат тот же самый числовой идентификатор.

Вновь вернитесь к методу CMainFrame::OnCreate() и вставьте операторы формирования этого элемента управления:

```
if( !m_check.Create("Выбрать здесь",
      WS_CHILD|WS_VISIBLE|BS_AUTOCHECKBOX,
      CRect(0,0,20,20), this, IDC_CHECK))
{
  TRACE0("Failed to create checkbox\n")
  return -1;      //Элемент не сформирована.
}
```

Последняя операция — вставка флажка в панель rebar:

```
m_rebar/AddBar(&m_check, "Панель", NULL,
      RBBS_BREAK|RBBS_GRIPPERALWAYS);
```

Метод AddBar() использует в качестве аргументов четыре параметра: указатель на объект включаемого элемента управления, текст, который нужно вывести около него, указатель на растровое изображение, которое будет использовано в качестве фона для нового элемента, и стиль панели, который формируется в виде комбинации символьных флагов из следующего набора:

- RBBS_BREAK — устанавливает элемент в новой строке, даже если для него есть место в конце существующей;
- RBBS_CHILDEDGE — устанавливает элемент напротив дочернего окна фрейма;
- RBBS_FIXEDBMP — предотвращает перемещение растрового изображения в случае, если пользователь вручную изменяет размеры элемента управления;
- RBBS_FIXEDSIZE — блокирует изменение размеров элемента пользователем;

- RBBS_GRIPPERALWAYS — обеспечивает вывод маркеров размеров;
- RBBS_HIDDEN — скрывает элемент;
- RBBS_NOGRIPPER — блокирует вывод маркеров изменения размеров;
- RBBS_NOVERT — скрывает элемент, если панель устанавливается вертикально;
- RBBS_VARIABLEHEIGHT — разрешает изменение размеров элементов при изменении размеров панели.

Теперь можно оттранслировать проект и запустить его на выполнение. В окне приложения вы должны увидеть новую панель, как показано на рис. 9.18. Проверьте, как реагирует флажок на щелчки мышью на нем. Он устанавливается или сбрасывается, но больше в программе ничего не происходит.

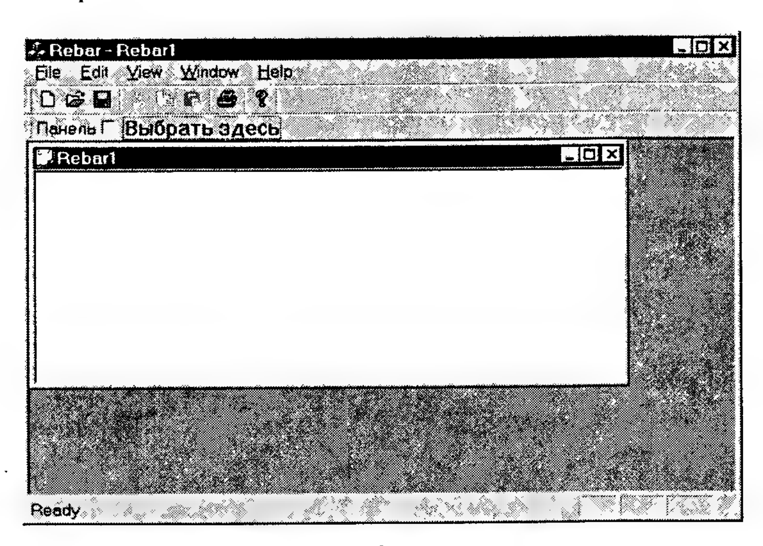


Рис. 9.18. В панель rebar включен флажок

Чтобы программа каким-то образом реагировала на установку или сброс флажка, нужно организовать перехват соответствующего сообщения и выполнить в ответ на него какое-либо действие. Проще всего изменить рисунок, формируемый методом OnDraw() класса представления. Щелкните дважды на имени класса CRebarView в окне CLassView и отредактируйте файл заголовка этого класса. Перейдите к карте сообщений и вставьте строку между замыкающим комментарием AFX_MSG и макросом DECLARE_MESSAGE_MAP().

```
afx_msg void OnClick();
```

Теперь разверните класс CRebarView в окне CLassView и дважды щелкните на имени функции OnDraw(). После текста этой функции в окне редактора кода добавьте

```
void CRebarView::OnClick();
{
   Invalidate();
}
```

Таким образом, при любом изменении состояния флажка изображение в окне будет перерисовываться. Найдите в начале файла карту сообщений и добавьте в нее строку (после трех элементов, касающихся распечатки):

```
ON_BN_CLICKED(IDC_CHECK, OnClick)
```

В самом начале файла после всех директив include добавьте еще одну строку:

```
#include "mainFrm.h"
```

А теперь в реализацию метода OnDraw() вместо комментариев TODO... вставьте следующий текст:

```
CString message;
if(((CMainFrame*)(AfxGetApp()->p_MainWnd))->m_check.GetCheck())
   message = "Флажок установлен";
else
   message = "Флажок сброшен";
pDC->TextOut(20,20,message);
```

В выражении if извлекается указатель на главное окно приложения. Он приводится к указателю на объект класса CMainFrame, и анализируется состояние флажка. Соответственно последнему формируется то или иное сообщение.

Теперь вновь оттранслируйте программу и запустите ее на выполнение. При каждом щелчке на флажке должно выводиться новое сообщение в окне приложения (рис. 9.19).

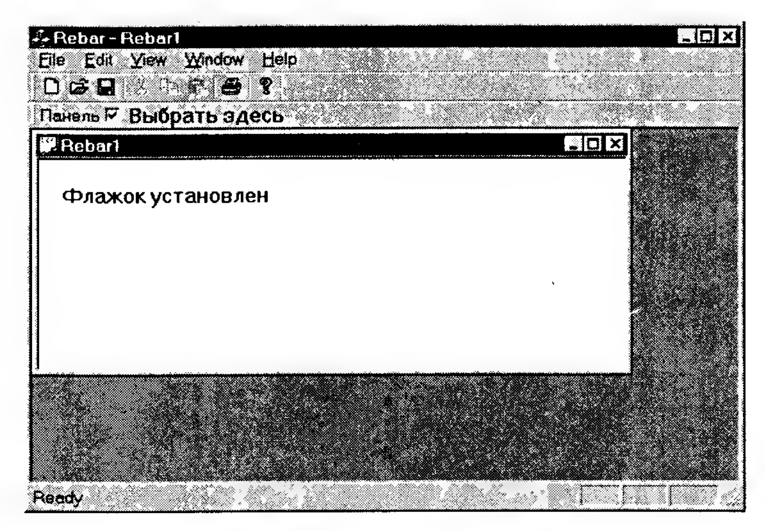


Рис. 9.19. Каждый щелчок на флажке, установленном в rebar, изменяет текст сообщения

ГЛАВА 10

Элементы управления общего назначения

В этой главе...

Линейный индикатор

Линейный регулятор

Инкрементный регулятор

Список изображений

Просмотровое окно списка

Просмотровое окно дерева

Расширенное текстовое поле

Элемент формирования IP-адреса

Элемент управления для работы с датами

Календарь

Прокрутка изображения

Для любого пользователя Windows привычными являются такие элементы управления, как кнопки, раскрывающиеся списки, меню и текстовые поля. Но по мере развития операционной среды Windows специалисты Microsoft с удивлением обнаружили, что изобретательные программисты не ограничились фирменными элементами и начали придумывать свои. Среди таких "доморощенных" средств оказались панели инструментов, строки состояния, линейные индикаторы, различные окна просмотра сложных иерархических структур данных и многое другое. Для того чтобы придать этому движению организованный характер и избавить энтузиастов от изобретения велосипедов, Microsoft включила элементы управления перечисленных типов в операционную среду Windows 95, а также в последние версии Windows NT и Windows 98. Теперь программисты, разрабатывающие продукты, ориентированные на Windows, могут заняться своим непосредственным делом, не отвлекаясь на конструирование собственных версий многофункциональных элементов управления. В этой главе вы познакомитесь со многими стандартными элементами управления, нашедшими применение в 32-разрядных приложениях Windows. Что касается панелей инструментов и строк состояния, то им посвящена глава 9. Средства просмотра свойств и мастера будут рассмотрены в главе 12.

В качестве примера в этой главе будет рассмотрено приложение Common. В нем используются девять многофункциональных элементов управления из набора средств Windows 95 — линейный индикатор (progress bar), линейный регулятор (slider или trackbar), инкрементный регулятор (up-down control или spinner), просмотровое окно списка (list view control), просмотровое окно дерева (tree view control), расширенное текстовое поле (reach edit control), элемент формирования адреса протокола Internet (IP-адреса), элемент для работы с датами (date picker) и календарь. Все они представлены на рис. 10.1. В следующих ниже разделах вы познакомитесь с основными методами формирования и использования в приложениях этих элементов.

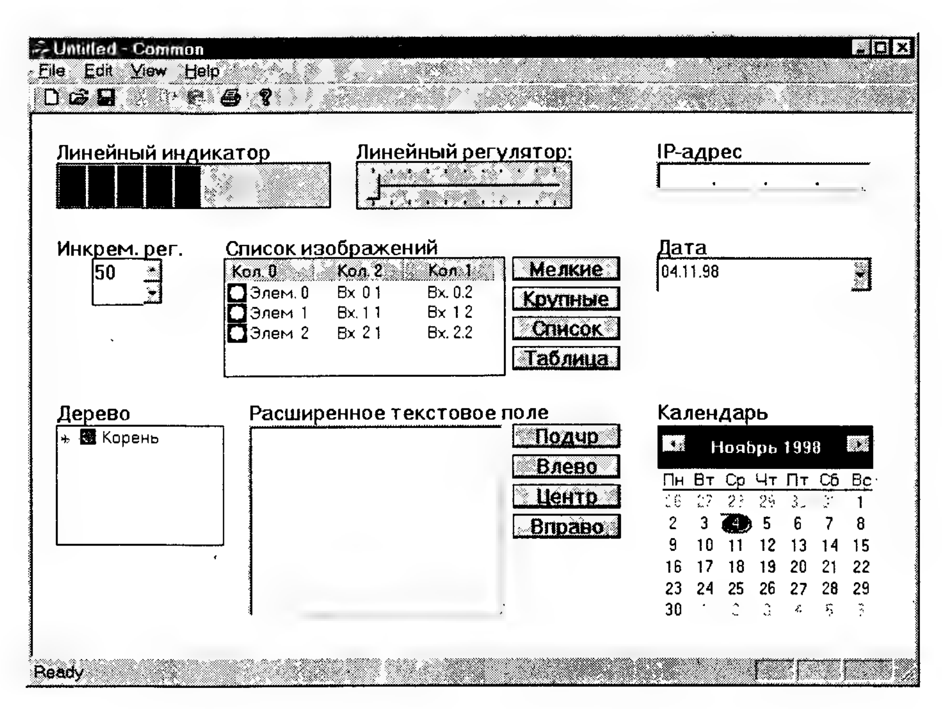


Рис. 10.1. Приложение Common демонстрирует многофункциональные элементы управления из набора средств Windows 95

Сформируйте заготовки приложения Common при помощи мастера AppWizard. На первом этапе установите создание приложения с единственным документном (SDI-приложения) и оставьте все предлагаемые мастером по умолчанию настройки вплоть до этапа 6. Здесь откройте раскрывающийся список Base Class и выберите в нем CScrollView. Таким образом в главное окно приложения будут автоматически установлены средства прокрутки изображения и пользователь сможет видеть все элементы управления, даже если они и не смогут уместиться одновременно на видимом поле экрана. После этого щелкните на Finish — и мастер сформирует заготовку приложения.

Элементы управления объявляются в качестве членов-переменных класса представления. Щелкните дважды на имени класса представления CCommonView в окне ClassView и отредактируйте файл заголовка этого класса — добавьте в него операторы объявления, представленные в листинге 10.1.

В следующем разделе мы подробно рассмотрим линейный индикатор, который представлен в классе переменной-объектом класса CProgressCtrl. Об остальных элементах речь пойдет в соответствующих разделах этой главы.

Листинг 10.1. Файл CommonView.h — объявление элементов управления

```
protected:
    // Линейный индикатор.
   CProgressCtrl m_progressBar;

    // Линейный регулятор.
   CSliderCtrl m_trackbar;
   BOOL m_timer;

    // Инкрементный регулятор.
   CSpinButtonCtrl m_upDown;
   CEdit m_buddyEdit;

    // Просмотровое окно списка.
   CListCtrl m_listView;
   CImageList m_smallImageList;
   CImageList m_largeImageList;
   CButton m_smallButton;
   CButton m_largeButton;
   CButton m_listButton;
   CButton m_reportButton;

    // Просмотровое окно дерева.
   CTreeCtrl m_treeView;
   CImageList m_treeImageList;

    // Расширенное текстовое поле.
   CRichEditCtrl m_richEdit;
   CButton m_boldButton;
   CButton m_leftButton;
   CButton m_centerButton;
   CButton m_rightButton;
     // IP-адрес.
   CIPAddressCtrl m_ipaddress;
     // Элемент работы с датами.
   CDateTimeCtrl m_date;
     // Календарь.
   CMonthCalCtrl m_month;
```

Разверните класс CCommonView и дважды щелкните на его методе OnDraw(). В тексте этой функции замените комментарий TODO... следующими операторами.

```
pDC->TextOut(20, 22, "Линейный индикатор");
pDC->TextOut(270, 22, "Линейный регулятор:");
pDC->TextOut(20, 102, "Инкрем. рег.");
pDC->TextOut(160, 102, "Список изображений");
pDC->TextOut(20, 240, "Дерево");
pDC->TextOut(180, 240, "Расширенное текстовое поле");
pDC->TextOut(520, 22, "IP-адрес");
pDC->TextOut(520, 102, "Дата");
pDC->TextOut(520, 240, "Календарь");
```

Эти надписи будут выведены в окне работающего приложения и помогут сориентироваться среди множества элементов управления на экране, которые мы будем добавлять по мере усложнения программы.

Линейный индикатор

Вероятно, самым простым из новых элементов является *линейный индикатор* (progress bar), который представляет собой прямоугольник, медленно заполняющийся цветными квадратиками. Чем больше заполнен прямоугольник, тем ближе завершение процесса. Когда прямоугольник заполнится окончательно, завершится и связанный с ним процесс. Этот элемент управления можно использовать в качестве индикатора хода выполнения некоторого процесса в приложении, например сортировки или загрузки длинного файла.

Формирование линейного индикатора

Прежде чем использовать линейный индикатор, его, естественно, нужно сформировать. В программах, ориентированных на библиотеку MFC, элементы управления зачастую формируются, как часть диалогового окна. Однако в приложении Common они демонстрируются непосредственно в главном окне, т.е. в представлении SDI-приложения. Вы можете более подробно познакомиться с документами и их представлениями в главе 4. Все элементы управления формируются методом OnCreate() класса представления CCommonView. Эта функция обрабатывает сообщение Windows WM_CREATE. Включить этот метод в класс представления можно при помощи уже не раз описанной методики. Щелкните правой кнопкой мыши на имени класса CCommonView в окне ClassView и выберите в контекстном меню пункт Add Windows Message Handler. Выберите WM_CREATE из списка слева и затем щелкните на Add and Edit. На место комментария TODO добавьте эту строку:

```
CreateProgressBar();
```

Снова щелкните правой кнопкой мыши на имени класса CCommonView в окне ClassView и на этот раз выберите в контекстном меню пункт Add Member Function. Задайте тип возвращаемого значения в поле Function Type как void, а в поле Function Declaration введите объявление функции CreateProgressBar(). Метод должен быть объявлен открытым (public). Теперь после щелчка на OK можно будет добавить в эту функцию код из листинга 10.2.

Листинг 10.2. Файл CommonView.cpp — функция CCommonView::CreateProgressBar()

```
void CCommonView::CreateProgressBar()
{
  m_progressBar.Create(WS_CHILD | WS_VISIBLE | WS_BORDER,
     CRect(20, 40, 250, 80), this, IDC_PROGRESSBAR);

  m_progressBar.SetRange(1, 100);
```

```
m_progressBar.SetStep(10);
m_progressBar.SetPos(50);
m_timer = FALSE;

}
```

Функция CreateProgressBar(), первым делом, формирует элемент управления, вызывая его метод Create() (если быть точным, то метод класса, связанного с элементом управления). Четыре аргумента функции имеют следующее назначение — флаги стиля отображения элемента, размер (как объект класса CRect), указатель родительского окна и идентификатор элемента управления. Идентификатор ресурса IDC_PROGRESSBAR добавляется вручную. Для того чтобы включить символы ресурсов в создаваемое приложение, выберите View⇨Resource Symbols и щелкните на кнопке New. После этого введите в соответствующее поле идентификатор ресурса (что-нибудь наподобие IDC_PROGRESSBAR) и используйте номер, который вам предлагает Visual Studio.

Константы стиля — это те же константы, которые используются для описания любого окна (ведь элемент управления есть ничто другое, как окно специального вида). В данном случае вам понадобятся, по крайней мере, следующие константы.

- WM_CHILD. Указывает, что элемент является дочерним окном.
- WM_VISIBLE. Предоставляет пользователю возможность видеть элемент на экране.

Кроме того, рекомендуется использовать еще и константу WM_BORDER, задающую специфический дизайн элемента на экране, при котором по контуру элемента появляется темная окантовка для выделения его на фоне окна.

Инициализация линейного индикатора

После того как элемент управления сформирован, его нужно инициализировать. Класс CProgressCtrl располагает рядом функций-членов, которые позволяют инициализировать элемент управления и манипулировать им. Эти функции перечислены в табл. 10.1.

Таблица 10.1. Функции-члены класса CProgressCtrl

Функция	Назначение
Create()	Создает элемент управления типа линейный индикатор
OffsetPos()	Продвигает зону заполнения на заданное количество блоков (квадратиков)
SetPos()	Устанавливает текущую величину параметра элемента
SetRange()	Устанавливает максимальное и минимальное значения параметра элемента
SetStep()	Устанавливает приращение параметра элемента, которое соответствует одному блоку
StepIt()	Добавляет один блок в зону заполнения

Для того чтобы инициализировать элемент управления, функция CCommonView::CreateProgressBar() последовательно вызывает SetRange(), SetStep()и SetPos(). Поскольку инкремент приращения и диапазон связаны, элемент, имеющий диапазон 1–10, и инкремент, равный 1, работают почти точно так же, как и элемент управления с диапазоном 1–100 и инкрементом 10.

Когда приложение только-только запущено, линейный индикатор заполнен наполовину цветными блоками. Это сделано сугубо из эстетических соображений. Обычно индикатор вначале пуст. Заполнение наполовину сделано функцией SetPos(), которая вызвана из CreateProgressBar()с аргументом 50, т.е. значение параметра равно половине диапазона.

Управление линейным индикатором

Обычно обновление состояния индикатора производится по мере того, как выполняемая задача близится к завершению. В приложении Common индикатор заполняется сигналами от таймера. Когда пользователь щелкает на свободном месте окна, программа запускает таймер, который передает сообщения WM_TIMER два раза в секунду. Нужно организовать перехват этого сообщения и обновление состояния индикатора. Делается это следующим образом.

1. Откройте ClassWizard. В раскрывающемся списке справа вверху выберите класс CCommonView.
2. В списке, который находится ниже справа, найдите сообщение WM_LBUTTONDOWN, которое вырабатывается при щелчке мышью на поле окна приложения, и выберите его.
3. Щелкните на кнопке Add Function, а затем — на кнопке Edit Code.
4. Отредактируйте текст функции OnLButtonDown(), который должен выглядеть следующим образом.

```
void CCommonView::OnLButtonDown(UINT nFlags, CPoint point)
{
    if (m_timer)
    {
        KillTimer(1);
        m_timer = FALSE;
    }
    else
    {
        SetTimer(1, 500, NULL);
        m_timer = TRUE;
    }
    CView::OnLButtonDown(nFlags, point);
}
```

Эта программа обеспечивает включение-выключение таймера после каждого щелчка кнопкой мыши. Параметр 500 при вызове SetTimer() задает интервал генерации сообщений WM_TIMER (размерность параметра — миллисекунды). В результате при такой настройке таймер будет формировать сообщение дважды в секунду.

5. На случай, если таймер будет продолжать работать и после закрытия окна представления, перегрузите метод OnDestroy() — вставьте оператор, запрещающий работу таймера. Для этого щелкните правой кнопкой мыши на строке CCommonView в окне ClassView и выберите в контекстном меню команду Add Windows Message Handler. Выберите WM_DESTROY и щелкните на Add and Edit. Вместо комментария TODO вставьте оператор

   ```
   KillTimer(1);
   ```

6. Теперь организуйте перехват сообщения таймера. Снова откройте ClassWizard и в списке, который находится справа, найдите сообщение WM_TIMER (строка этого сообщения находится почти в самом конце списка, упорядоченного по алфавиту). Щелкните на кнопке Add Function, а затем — на кнопке Edit Code. Вместо комментария TODO вставьте оператор

   ```
   m_progressBar.StepIt();
   ```

Функция StepIt()увеличивает параметр элемента управления на величину, определенную функцией SetStep() при инициализации, и, таким образом, добавляет один блок в зону заполнения индикатора. Когда параметр элемента достигнет максимума, он автоматически сбросится и заполнение начнется сначала.

На заметку Учтите, что среди функций-членов класса CProgressCtrl нет таких, которые явно задавали бы размер блока и максимальное количество блоков, которые можно добавить в индикатор. Эти атрибуты задаются неявно через размеры элемента управления.

Теперь оттранслируйте приложение и посмотрите, как работает в нем линейный индикатор.

Линейный регулятор

Во время работы программ нередко возникают ситуации, когда пользователь должен ввести некоторую величину, не выходящую за пределы заданного диапазона. Для таких задач можно использовать класс CSliderCtrl из библиотеки MFC. С его помощью можно сформировать элемент управления типа линейный регулятор (slider или trackbar). Предположим, пользователю нужно ввести процентное соотношение. В этом случае предполагается, что введенная величина будет находиться в диапазоне от 0 до 100. Если будет введена другая величина, это может привести к определенным сбоям в программе (или ее придется специально ограничивать).

Если же использовать линейный регулятор, то пользователь даже при большом желании не сможет сделать ошибку.

Для случая ввода процентного отношения нижний предел входной величины равен 0, а верхний — 100. Более того, для облегчения работы можно "разметить" элемент пометками, которые соответствуют десяткам (для этого потребуется 11 пометок). Именно такой элемент управления и включен в приложение Common.

Чтобы увидеть линейный регулятор в работе, нужно щелкнуть на "щели" регулятора. Когда вы это сделаете, бегунки регулятора сдвинутся в сторону того участка щели, на котором произошел щелчок, а соответствующее значение параметра появится справа от заголовка элемента управления, как это показано на рис. 10.1. Если регулятор имеет *фокус ввода*, им можно управлять и с помощью клавиш управления курсором <↑> и <↓>, а также <PgUp> и <PgDn>.

Создание линейного регулятора

Для нового элемента управления нам понадобится соответствующий идентификатор ресурса, который можно сформировать по той методике, которая была продемонстрирована ранее. Выберите View⇨Resource Symbols и щелкните на кнопке New. После этого введите в соответствующее поле идентификатор ресурса (в нашем случае — IDC_TRACKBAR) и используйте то числовое значение, которое предлагает Visual Studio. В метод CCommonView::OnCreate() добавьте вызов функции CreateTrackBar(). Затем добавьте в этот класс новый метод CreateTrackBar(), текст которого представлен в листинге 10.3.

Листинг 10.3. Файл CommonView.cpp — функция CCommonView::CreateTrackBar()

```
void CCommonView::CreateTrackbar()
{
  m_trackbar.Create(WS_CHILD | WS_VISIBLE | WS_BORDER |
    TBS_AUTOTICKS | TBS_BOTH | TBS_HORZ,
    CRect(270, 40, 450, 80), this, IDC_TRACKBAR);
  m_trackbar.SetRange(0, 100, TRUE);
  m_trackbar.SetTicFreq(10);
  m_trackbar.SetLineSize(1);
  m_trackbar.SetPageSize(10);
}
```

Как и в случае с линейным индикатором, функция CreateTrackBar() сначала формирует элемент управления, вызывая его метод Create(). Четыре аргумента функции имеют следующее назначение — флаги стиля отображения элемента, размер (как объект класса CRect), указатель родительского окна и идентификатор элемента управления. Флаги стиля задаются как комбинация символов констант стиля, в набор которых входят как константы, применимые для всех окон, так и специально созданные для линейного регулятора. В табл. 10.2 представлены эти специальные константы стиля.

Таблица 10.2. Константы стиля для линейного регулятора

Константа	Назначение
TBS_AUTOTICKS	Разрешает автоматическое формирование пометок
TBS_BOTH	Рисует пометки по обе стороны от щели регулятора
TBS_BOTTOM	Рисует пометки ниже щели горизонтального регулятора
TBS_ENABLESELRANGE	Позволяет выводить поддиапазоны регулятора
TBS_HORZ	Рисует горизонтальный регулятор
TBS_LEFT	Рисует пометки слева от щели вертикального регулятора
TBS_NOTICKS	Рисует регулятор без пометок
TBS_RIGHT	Рисует пометки справа от щели вертикального регулятора
TBS_TOP	Рисует пометки над щелью горизонтального регулятора
TBS_VERT	Рисует вертикальный регулятор

Инициализация линейного регулятора

После того как линейный регулятор создан, его нужно инициализировать. Класс CSliderCtrl располагает рядом функций-членов, которые позволяют инициализировать элемент управления этого типа и манипулировать им. Данные функции перечислены в табл. 10.3.

Таблица 10.3. Функции-члены класса CSliderCtrl

Функция	Назначение
ClearSel()	Очищает выполненную ранее установку регулятора
ClearTics()	Очищает установленные пометки регулятора
Create()	Создает элемент управления типа линейный регулятор
GetChannelRect()	Считывает размер регулятора элемента управления
GetLineSize()	Считывает установленное значение приращения параметра элемента управления в ответ на нажатие клавиши управления курсором <↑> или <↓>
GetNumTics()	Считывает количество пометок элемента управления
GetPageSize()	Считывает размер страницы элемента управления (значение приращения параметра при управлении клавишами <PgUp> и <PgDn>)
GetPos()	Считывает текущее значение параметра регулятора элемента управления
GetRange()	Считывает диапазон значений параметра элемента управления (максимальное и минимальное значения)

Функция	Назначение
GetRangeMax()	Считывает максимальное значение диапазона допустимых значений параметра элемента управления
GetRangeMin()	Считывает минимальное значение диапазона допустимых значений параметра элемента управления
GetSelection()	Считывает текущий выбор диапазона допустимых значений параметра элемента управления
GetThumbRect()	Считывает размер прямоугольника бегунка элемента управления
GetTic()	Считывает положение ближайшей пометки элемента управления
GetTicArray()	Считывает положения всех пометок элемента управления
GetTicPos()	Считывает координаты пометки элемента управления
SetLineSize()	Устанавливает значение приращения параметра элемента управления в ответ на нажатие клавиши управления курсором <↑> или <↓>
SetPageSize()	Устанавливает размер страницы элемента управления (значение приращения параметра при управлении клавишами <PgUp> и <PgDn>)
SetPos()	Устанавливает положение регулятора элемента управления
SetRange()	Устанавливает диапазон значений параметра элемента управления (максимальное и минимальное значения)
SetRangeMax()	Устанавливает максимальное значение диапазона допустимых значений параметра элемента управления
SetRangeMin()	Устанавливает минимальное значение диапазона допустимых значений параметра элемента управления
SetSelection()	Устанавливает поддиапазон допустимых значений параметра элемента управления
SetThumbRect()	Устанавливает размер прямоугольника бегунка элемента управления
SetTic()	Устанавливает положение пометки элемента управления
SetTicFreq()	Устанавливает интервал между пометками элемента управления
VerifyPos()	Проверяет, допустимо ли заданное пользователем значение параметра

Как правило, если уж вы включили в свое приложение линейный регулятор, то должны установить и диапазон значений соответствующего параметра, и интервал между отсчетами (пометками на изображении регулятора). Кроме того, если предусматривается возможность управления регулятором с помощью клавиатуры, необходимо установить значение *линейного* приращения параметра элемента управления в ответ на нажатие клавиши управления курсором <↑> или <↓> и *размер страницы* элемента управления — значение приращения параметра при управлении клавишами <PgUp> и <PgDn>. В приложении Common линейный регулятор инициализируется функциями SetRange(), SetTicFreq(), SetLineSize() и SetPageSize(), как это показано в листинге 10.3. При вызове SetRange() значения верхнего и нижнего пределов диапазона устанавливаются равными соответственно 0 и 100. Третий аргумент вызова этой функции (типа BOOL) задает, нужно ли перерисовывать изображение регулятора после изменения заданного диапазона. Обратите внимание, что значения интервала между пометками (в единицах значений параметра) и *размер страницы* равны. Это необязательно, но крайне желательно для нормальной работы с элементом при использовании клавиатуры. Пользователь будет неприятно удивлен, если, нажав клавишу <PgUp> или <PgDn>, увидит, что бегунок регулятора убежал куда-то в конец шкалы.

Другие методы класса CSliderCtrl позволяют менять размер изображения элемента управления на экране, размер бегунка, задавать значение параметра регулятора и выполнять множество других функций.

Работа с линейным регулятором

Линейный регулятор — это не что иное как разновидность полосы прокрутки. Когда пользователь перемещает бегунок регулятора, элемент управления формирует сообщение WM_HSCROLL, перехват которого в классе представления нам и предстоит организовать. Откройте окно ClassWizard, а в нем — вкладку Message Maps и выберите в правом верхнем списке CCommonView, а в списке сообщений — WM_HSCROLL. Щелкните на кнопке Add Function, а затем — на кнопке Edit Code. Введите текст листинга 10.4.

Листинг 10.4. Файл CommonView.cpp — функция CCommonView::OnHScroll()

```
void CCommonView::OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
  CSliderCtrl* slider = (CSliderCtrl*)pScrollBar;
  int position = slider->GetPos();
  char s[10];
  wsprintf(s, "%d  ", position);
  CClientDC clientDC(this);
  clientDC.TextOut(390, 22, s);

  CScrollView::OnHScroll(nSBCode, nPos, pScrollBar);
}
```

Из текста программы следует, что сам по себе элемент управления не выводит где-либо значения параметра. Это берет на себя функция OnHScroll(). Ниже приведена последовательность действий, выполняемых при этом.

1. Четвертый аргумент функции OnHScroll()представляет собой указатель на объект — элемент управления, подобный полосе прокрутки, который формирует сообщение WM_HSCROLL.
2. Сначала этот указатель приводится к типу класса CSliderCntr, а затем он служит для считывания текущего значения параметра регулятора с помощью метода GetPos() класса CSliderCntr.
3. Получив значение параметра (типа int), функция преобразует его в строку и выводит ее на экран.

Подробно процесс вывода текста на экран рассмотрен в главе 5. Прежде чем перейти к следующему типу элемента управления, оттранслируйте приложение и проверьте, как работает только что установленный элемент.



Если в настройках Windows выбран режим работы с крупными шрифтами, текущее значение, заданное для регулятора, может выводиться не там, где вы предполагали. Это происходит вследствие того, что надпись *Trackbar Control* занимает слишком много места из-за использования крупного шрифта. Если такое случится, откорректируйте аргументы вызова TextOut() с тем, чтобы значение выводилось немного правее.

Инкрементный регулятор

Линейный регулятор — это далеко не единственное средство, которое можно использовать в приложении для того, чтобы принять от пользователя значение некоторого параметра в заданном диапазоне. Если для вас не очень важна визуальная обратная связь, которую обеспечивает линейный регулятор, можете воспользоваться *инкрементным регулятором* (up-down control). На экране он выглядит, как пара стрелок, на которых пользователь может щелкать мышью, увеличивая или уменьшая значение параметра. Как правило, рядом с таким регулятором в экранной форме размещается другой элемент управления — текстовое поле (его часто называют *полем-отростком* — buddy control), в котором выводится числовое значение параметра.

В приложении Common параметр, который связан с инкрементным регулятором, можно изменять, просто щелкая на его стрелках. При этом в расположенном рядом поле значение меняется, показывая до чего пользователь "дорегулировался". Если регулятор получил фокус ввода, то параметр можно менять и с помощью клавиш управления курсором <↑> и <↓>.

Формирование инкрементного регулятора

Добавьте в CCommonView::DnCreate() вызов еще одной функции —.CreateUpDownCtrl(), текст которой представлен в листинге 10.5. Добавьте также новые идентификаторы ресурсов — IDC_BUDDYEDIT и IDC_UPDDWN.

Листинг 10.5. Файл CommonView.cpp — функция CCommonView::CreateUpDownCtrl()

```
void CCommonView::CreateUpDownCtrl()
{
  m_buddyEdit.Create(WS_CHILD | WS_VISIBLE | WS_BDRDER,
    CRect(50, 120, 110, 160), this, IDC_BUDDYEDIT);
  m_upDown.Create(WS_CHILD | WS_VISIBLE | WS_BDRDER |
    UDS_ALIGNRIGHT | UDS_SETBUDDYINT | UDS_ARRDWKEYS,
    CRect(0, 0, 0, 0), this, IDC_UPDDWN);
  m_upDown.SetBuddy(&m_buddyEdit);
  m_upDown.SetRange(1, 100);
  m_upDown.SetPos(50);
}
```

При создании инкрементного регулятора эта функция формирует связанный с ним элемент управления — отросток (buddy control), в котором будет выводиться значение параметра. В большинстве случаев, включая и даниый, элементом-отростком является текстовое поле. Для его создания вызывается функция-член Create() класса CEdit. Четыре аргумента функции имеют следующее назначение — флаги стиля отображения элемента, размер элемента управления, указатель родительского окна и идентификатор элемента управления. Если вы вернетесь к объявлению элементов управления приложения, то убедитесь, что m_buddyEdit является экземпляром класса CEdit.

После формирования элемента-отростка программа уже может приступить к формированию собственно инкрементного регулятора по той же методике — вызывается функция-член Create() класса, соответствующего этому типу элементов управления. Как вы уже, наверное, догадались, функция имеет четыре аргумента — флаги стиля отображения элемента, размер элемента управления, указатель родительского окна и идентификатор элемента управления. Как и для большинства элементов управления, флаги стиля задаются в виде комбинации символов констант стиля, в набор которых входят как константы, применимые для всех окон, так и специально объявленные для инкрементного регулятора в классе CSpinButtonCtrl. В табл. 10.4 представлены эти специальные константы стиля.

Таблица 10.4. Константы стиля для инкрементного регулятора

Стиль	Назначение
UDS_ALIGNLEFT	Размещает регулятор у левой границы элемента-отростка
UDS_ALIGNRIGHT	Размещает регулятор у правой границы элемента-отростка
UDS_ARROWKEYS	Позволяет пользователю применять клавиши управления курсором <↑> и <↓> для изменения параметра регулятора
UDS_AUTOBUDDY	Размещает на экране последовательно элемент-отросток и регулятор
UDS_HORZ	Рисует горизонтвльный регулятор
UDS_NOTHOUSANDS	Удаляет разделитель между триадами цифр
UDS_SETBUDDYINT	Выводит значение параметра регулятора в элементе-отростке
UDS_WRAP	Устанавливает режим "кольца", когда после достижения верхнего предела диапазона параметр получает значение нижнего предела

После того как инкрементный регулятор сформирован, его необходимо инициализировать. Класс CSpinButtonCtrl располагает набором функций-членов, которые позволяют инициализировать эти элементы управления и манипулировать ими. Функции приведены в табл. 10.5.

Таблица 10.5. Функции-члены класса CSpinButtonCtrl

Функция	Назначение
Create()	Создает элемент управления типа инкрементный регулятор
GetAccel()	Считывает скорость изменения параметра
GetBase()	Считывает основание системы счисления для вывода значения параметра
GetBuddy()	Считывает указатель объекта — элемента управления-отростка
GetPos()	Считывает текущее значение параметра регулятора
GetRange()	Считывает диапазон значений параметра элемента управления (максимальное и минимальное значения)
SetAccel()	Устанавливает скорость изменения параметра
SetBase()	Устанавливает основание системы счисления для вывода значения параметра (10 — для десятичной системы, 16 — для шестнадцатеричной)
SetBuddy()	Устанавливает указатель объекта — элемента управления-отростка
SetPos()	Устанавливает текущее значение параметра регулятора
SetRange()	Устанавливает диапазон значений параметра элемента управления (максимальное и минимвльное значения)

Приложение, которое рассматривается в этой главе, используется для инициализации инкрементного регулятора функций SetBuddy(), SetRange() и SetPos(). Поскольку в Create() передается в качестве одного из аргументов флаг UDS_SETBUDDYINT и при инициализации вызывается функция-член SetBuddy(), приложение Common не нуждается ни в каких дополнительных настройках для того, чтобы на экране появилось значение регулируемого параметра. Элемент управления автоматически выполняет все операции, необходимые для управления своим "отростком". Оттранслируйте приложение и опробуйте новый элемент в работе.

Можно изменить настройки элемента с тем, чтобы отсчет значений выполнялся с меньшей или большей скоростью, или установить вывод значений в шестнадцатеричной системе вместо десятичной. Просмотрите список методов класса этого элемента управления в электронной справочной системе Visual C++.

Список изображений

В программных продуктах зачастую необходимо использовать простенькие изображения, некоторым образом связанные с функционированием программы. Типичным примером являются всем известные панели инструментов, в которых используются растровые изображения для представления пиктографических кнопок. Для подобных случаев желательно иметь в своем распоряжении программный объект, который был бы способен не только хранить такие растровые изображения, но и некоторым образом организовывать обращение к ним. Это именно то, что может делать элемент управления типа список изображений: он хранит список взаимосвязанных изображений. Разработчик может использовать эти изображения так, как он сочтет нужным, соответственно задаче, решаемой в создаваемой программе. Несколько элементов управления из числа используемых в Windows 95 связаны тем или иным образом со списками изображений. К ним относятся следующие.

- Просмотровое окно списка
- Просмотровое окно дерева
- Страница свойств
- Панель инструментов

Вы, несомненно, найдете таким элементам не одно полезное применение. Например, вам может понадобиться некоторая последовательность быстро сменяемых на экране изображений, которая создает эффект анимации. Прекрасным хранилищем для отдельных кадров послужит список пиктограмм, к которым можно будет обращаться по индексу.

Если термин *индекс* сразу вызвал у вас ассоциацию с массивом, значит, вы правильно поняли, как именно хранятся изображения в этом списке. Это действительно массив, только в отличие от привычных для всех программистов (и любителей, и профессионалов) чисел (целых или с плавающей запятой) он хранит изображения. Так же, как и в обычном массиве, нужно инициализировать каждый его элемент, а затем обращаться к нему, используя индекс.

Однако вам не удастся наяву увидеть такой список изображений в приложении, подобно тому, как вы видите на экране строку состояния или линейный индикатор. Причина в том, что список изображений есть не что иное как "склад" изображений (опять же, здесь полная аналогия с обычным массивом чисел). Можно вывести на экран изображения из массива, но нельзя отобразить сам массив. На рис. 10.2 показано, как организован список изображений.



Рис. 10.2. Список изображений во многом напоминает массив картинок

Формирование списка изображений

В приложении `Common` список изображений используется в элементах управления типа просмотровое окно списка и просмотровое окно дерева. Сам пиктографический список создается локальными методами `CreateListView()`и `CreateTreeView()`, которые вызываются функцией `CCommonView::OnCreate()`. Вскоре вы увидите их тексты, но они достаточно объемны, в этом разделе мы рассмотрим только два вызова функций, которые имеют отношение к

списку изображений. Список создается путем его обращения к методу Create(), как, например, здесь:

```
m_smallImageList.Create(16, 16, FALSE, 1, 0);
m_largeImageList.Create(32, 32, FALSE, 1, 0);
```

Функция Create() имеет пять аргументов.

- Ширина картинки
- Высота картинки
- Булева переменная, которая служит индикатором того, является ли данное изображение маской
- Начальное количество изображений в списке
- Количество изображений, на которое список может динамически увеличиться

Значение 0 последнего аргумента означает, что список не может динамически увеличиваться во время выполнения программы. Функция Create() перегружена в классе CImageList, так что можно создавать список разными способами. Другие версии функции Create() можно найти в оперативной документации на Visual C++.

Инициализация списка изображений

После создания списка нужно будет заполнить его изображениями. Ведь, в конце концов, пустой список просто не имеет смысла. Самый простой способ — включить изображения в состав файла ресурсов приложения и затем загрузить их оттуда. Например, следующий фрагмент текста программы показывает, как приложение Common загружает пиктограмму в один из списков изображений.

```
HICON hIcon = ::LoadIcon (AfxGetResourceHandle(), MAKEINTRESOURCE(IDI_ICON1));
m_smallImageList.Add(hIcon);
hIcon = ::LoadIcon (AfxGetResourceHandle(), MAKEINTRESOURCE(IDI_ICON2));
m_largeImageList.Add(hIcon);
```

Сначала программа получает дескриптор (handle) пиктограммы. Затем добавляет эту пиктограмму в список изображений, вызывая его (списка) функцию-член Add(). Само изображение пиктограммы можно создать, вызвав Insert⇨Resource, после чего сделать двойной щелчок на Icon и отредактировать изображение чистой пиктограммы, пользуясь Resource Editor (редактором ресурсов). При этом новый ресурс автоматически получит идентификатор IDI_ICON1. Теперь щелкните на пиктограмме New Device Image рядом с раскрывающимся списком и выберите в появившемся диалоговом окне Small(16×16). Щелкните на OK и закройте диалоговое окно. Для создания новой пиктограммы придется потратить немного времени, если есть желание нарисовать красивую картинку, но можно поступить проще — создать пиктограмму в виде белого круга на черном фоне. Для этого понадобится единственное обращение к инструменту Ellipse. После этого добавьте еще одно изображение пиктограммы, на сей раз размером 32x32. Этой пиктограмме будет присвоен идентификатор ресурса IDI_ICON2. Картинку на пиктограмме можно построить тем же способом.

Для работы с объектами класса CImageList можно использовать множество включенных в состав класса методов, которые позволяют настраивать цвета, добавлять и удалять изображения в список и т.д. Перечень этих методов вы найдете в системе электронной справки Visual C++ и в табл. 10.6.

Теперь включим в функцию CreateTreeView() несколько операторов, и будет создан один список изображений, в котором поначалу будут содержаться три элемента.

```
m_treeImageList.Create(13, 13, FALSE, 3, 0);
HICON hIcon = ::LoadIcon (AfxGetResourceHandle(), MAKEINTRESOURCE(IDI_ICON3));
m_treeImageList.Add(hIcon);
hIcon = ::LoadIcon (AfxGetResourceHandle(), MAKEINTRESOURCE(IDI_ICON4));
m_treeImageList.Add(hIcon);
hIcon = ::LoadIcon (AfxGetResourceHandle(), MAKEINTRESOURCE(IDI_ICON5));
m_treeImageList.Add(hIcon);
```

Три пиктограммы — IDI_ICON3, IDI_ICON4 и IDI_ICON5 — формируются так же, как первые две пиктограммы. Все три должны иметь размер 32×32. Как и ранее, пусть на пиктограммах будут изображены круги, но для разнообразия можно изменить цветовую гамму.

Таблица 10.6. Функции-члены класса CImageList

Функция	Назначение
Add()	Добавляет изображение в список
Attach()	Добавляет существующий список изображений к объекту класса CImageList
BeginDrag()	Запускает операцию "перетащить" изображения
Create()	Создает элемент управления типа список изображений
DeleteImageList()	Стирает список изображений
Detach()	Исключает список изображений из объекта класса CImageList
DragEnter()	Блокирует обновление окна и показывает "перетянутое" изображение
DragLeave()	Снимает блокировку обновления окна
DragMove()	Передвигает "перетаскиваемое" изображение
DragShowNoLock()	Управляет "перетаскиваемым" изображением без блокировки обновления окна
Draw()	Обрисовывает "перетаскиваемое" изображение
EndDrag()	Завершает операцию "перетащить" изображения
Extraction()	Создает пиктограмму из изображения
GetBkColor()	Считывает цвет фона списка изображений
GetDragImage()	Считывает изображение для операции "перетащить"
GetImageCount()	Считывает количество изображений, помещенных в элемент управления
GetImageInfo()	Считывает информацию об изображении
GetSafeHandle()	Считывает дескриптор списка изображений
Read()	Считывает список изображений из заданного архива
Remove()	Удаляет изображение из списка изображений
Replace()	Заменяет одно изображение другим
SetBkColor()	Устанавливает цвет фона списка изображений
SetDragCursorImage()	Формирует изображение для операции "перетащить"
SetOverlayImage()	Устанавливает индекс маски наложения
Write()	Записывает список изображений в заданный архив

Просмотровое окно списка

Элемент управления типа просмотровое окно списка упрощает разработку приложений, которые работают со списками объектов и организуют эти объекты таким образом, чтобы пользователь мог легко определить их атрибуты. Типичный пример — группа дисковых файлов. Каж-

дый файл представляет собой отдельный объект, который характеризуется множеством атрибутов — именем, размером, датой последней модификации и т.п. Windows 95 показывает их либо в виде пиктограмм в пределах окна, либо в виде таблицы, каждая колонка которой соответствует определенному атрибуту. Пользователю предоставлены возможности выбора формата отображения списка файлов, включая выбор отображаемых атрибутов. В набор стандартных элементов управления Windows 95 включено так называемое *просмотровое окно списка* (list view control), которое позволяет программисту организовать отображение необходимых в приложении списков точно в таком же формате, как и в фирменных программах.

Чтобы увидеть, что представляет собой "заполненное до краев" просмотровое окно списка, достаточно открыть Explorer Windows 95 (рис. 10.3). В правой части окна этой программы видно, как можно организовать вывод информации об объектах. (В левой части содержится просмотровое окно дерева, о котором речь пойдет в следующем разделе.) На этом рисунке просмотровое окно списка выведено в формате таблицы, в котором каждому элементу списка соответствует отдельная строка. В ней показано не только имя файла, но и связанные с ним атрибуты.

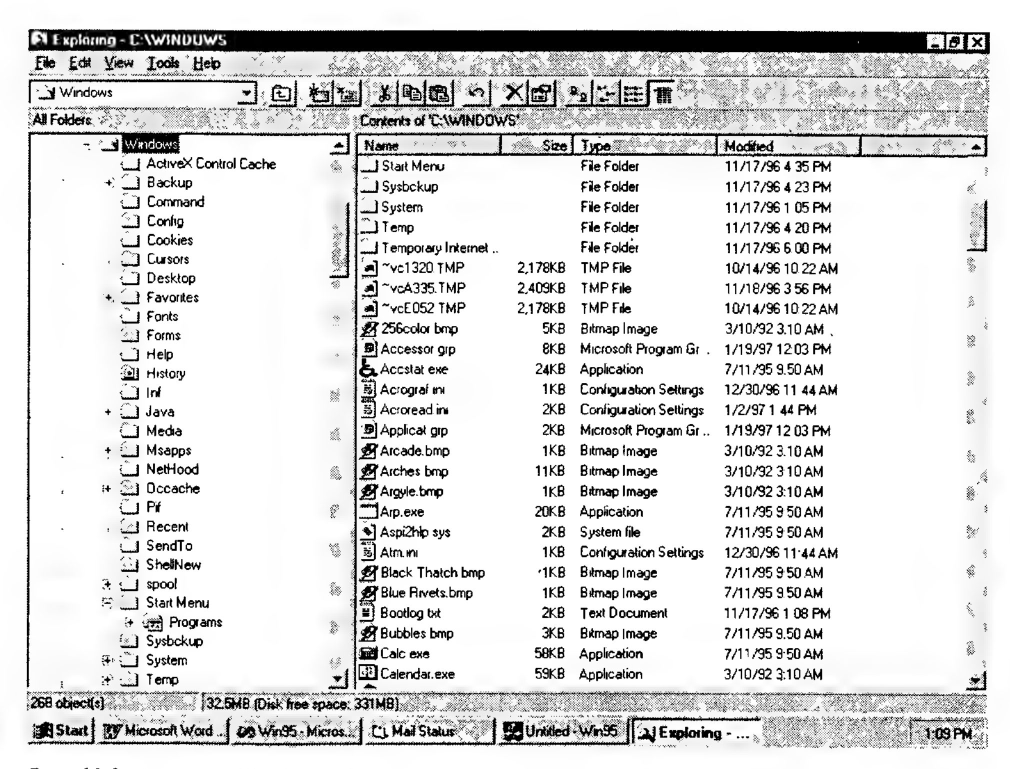


Рис. 10.3. Explorer Windows 95 использует просмотровое окно списка для организации информации о файлах

Пользователь может изменить формат представления файлов в окне. На рис. 10.4, например, показано, как будет выглядеть список файлов, если выбран формат больших пиктограмм (large icons), а на рис. 10.5 — формат малых пиктограмм (small icons). В последнем случае на экране умещается гораздо больше элементов списка. Пользуясь средствами просмотрового окна списка, пользователь может менять имена объектов — элементов списка, а в формате таблицы сортировать элементы на основе содержимого выбранных колонок.

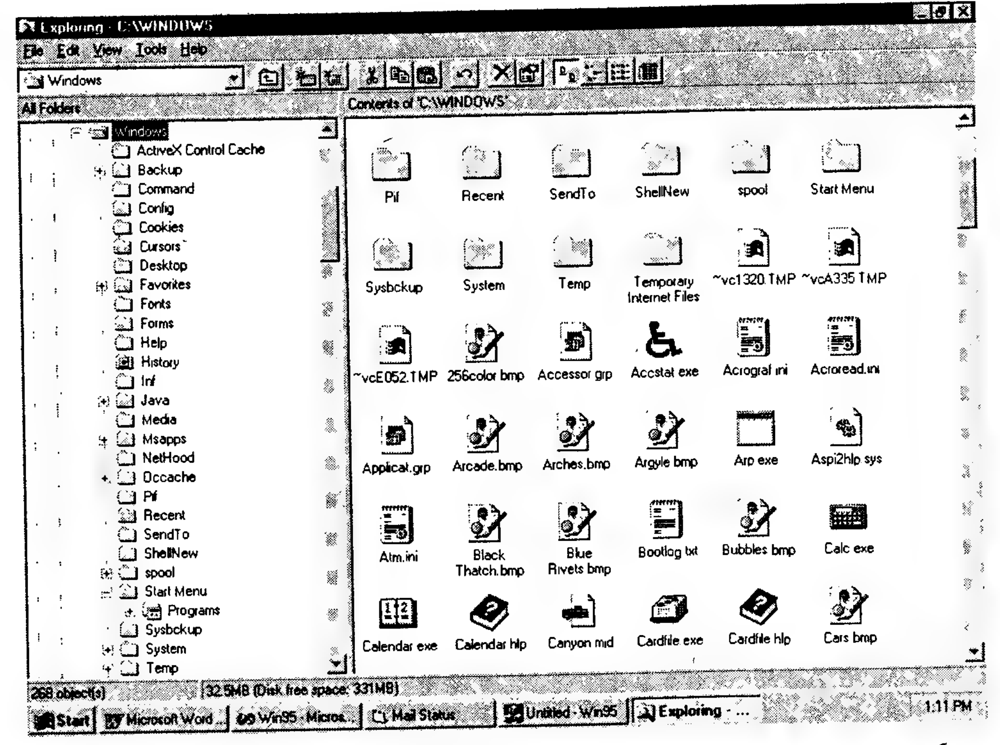


Рис. 10.4. Просмотровое окно списка Explorer *Windows 95, настроенное на формат больших пиктограмм*

Приложение Common также обращается к услугам просмотрового окна списка, хотя в данном случае решаются задачи поскромнее и все выглядит не так красиво, как в Explorer.

Создание просмотрового окна списка

Как же все это происходит? Возни с просмотровым окном списка гораздо больше, чем с линейным индикатором, линейным или инкрементным регулятором (трудно себе представить, что может быть легче, чем работа с этими элементами). В приложении Common просмотровое окно списка создается локальным методом CreateListView() класса CCommonView (листинг 10.6). Вам нужно самостоятельно сформировать и ввести текст этой функции, пользуясь редактором программного кода. Метод CreateListView() вызывается методом OnCreate() этого же класса. CreateListView() выполняет следующие действия.

1. Создает объекты — элементы управления типа список изображений.
2. Создает объект — элемент управления типа просмотровое окно списка.
3. Связывает списки изображений с просмотровым окном списка.
4. Формирует колонки.
5. Заполняет колонки необходимыми данными.
6. Создает элементы списка.
7. Заполняет элементы данными.
8. Создает кнопки управления форматом.

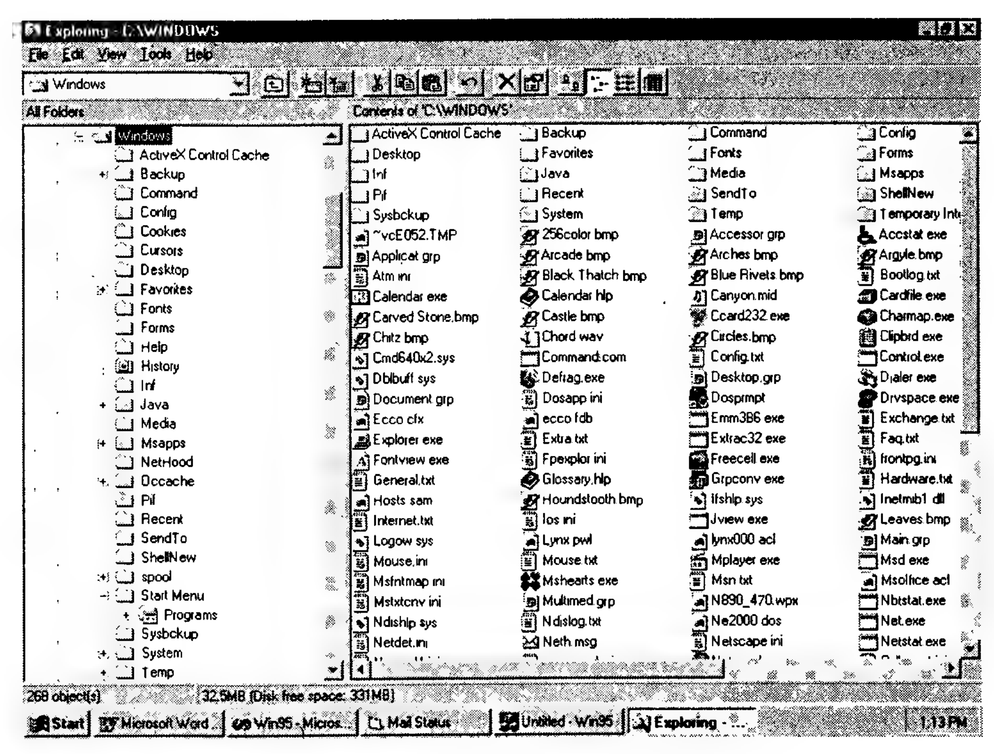


Рис. 10.5. Просмотровое окно списка Explorer *Windows 95, настроенное на формат малых пиктограмм*

Листинг 10.6. Файл `CommonView.cpp` **— функция** `CCommonView::CreateListView()`

```
void CCommonView::CreateListView()
{
// Создание пиктографического окна списка.
   m_smallImageList.Create(16, 16, FALSE, 1, 0);
   m_largeImageList.Create(32, 32, FALSE, 1, 0);

   HICON hIcon = ::LoadIcon (AfxGetResourceHandle(),
      MAKEINTRESOURCE(IDI_ICON1));
   m_smallImageList.Add(hIcon);
   hIcon = ::LoadIcon (AfxGetResourceHandle(),
       MAKEINTRESOURCE(IDI_ICON2));
   m_largeImageList.Add(hIcon);

   // Create the List View control.
   // Создание просмотрового  окна списка.
   m_listView.Create(WS_VISIBLE | WS_CHILD | WS_BORDER |
      LVS_REPORT | LVS_NOSORTHEADER | LVS_EDITLABELS,
      CRect(160, 120, 394, 220), this, IDC_LISTVIEW);

   m_listView.SetImageList(&m_smallImageList, LVSIL_SMALL);
   m_listView.SetImageList(&m_largeImageList, LVSIL_NORMAL);

   // Create the columns.
   // Формирование колонок.
```

```
    LV_COLUMN lvColumn;
    lvColumn.mask = LVCF_FMT | LVCF_WIDTH | LVCF_TEXT | LVCF_SUBITEM;
    lvColumn.fmt = LVCFMT_CENTER;
    lvColumn.cx = 75;

    lvColumn.iSubItem = 0;
    lvColumn.pszText = "Кол. 0";
    m_listView.InsertColumn(0, &lvColumn);
    lvColumn.iSubItem = 1;
    lvColumn.pszText = "Кол. 1";
    m_listView.InsertColumn(1, &lvColumn);
    lvColumn.iSubItem = 2;
    lvColumn.pszText = "Кол. 2";
    m_listView.InsertColumn(1, &lvColumn);

    // Create the items.
    // Формирование элементов списка.
    LV_ITEM lvItem;
    lvItem.mask = LVIF_TEXT | LVIF_IMAGE | LVIF_STATE;
    lvItem.state = 0;
    lvItem.stateMask = 0;
    lvItem.iImage = 0;

    lvItem.iItem = 0;
    lvItem.iSubItem = 0;
    lvItem.pszText = "Элем. 0";
    m_listView.InsertItem(&lvItem);
    m_listView.SetItemText(0, 1, "Вх. 0.1");
    m_listView.SetItemText(0, 2, "Вх. 0.2");

    lvItem.iItem = 1;
    lvItem.iSubItem = 0;
    lvItem.pszText = "Элем. 1";
    m_listView.InsertItem(&lvItem);
    m_listView.SetItemText(1, 1, "Вх. 1.1");
    m_listView.SetItemText(1, 2, "Вх. 1.2");

    lvItem.iItem = 2;
    lvItem.iSubItem = 0;
    lvItem.pszText = "Элем. 2";
    m_listView.InsertItem(&lvItem);
    m_listView.SetItemText(2, 1, "Вх. 2.1");
    m_listView.SetItemText(2, 2, "Вх. 2.2");

    // Create the view-control buttons.
    // Формирование кнопок управления просмотром.
    m_smallButton.Create("Мелкие", WS_VISIBLE | WS_CHILD | WS_BORDER,
        CRect(400, 120, 490, 140), this, IDC_LISTVIEW_SMALL);
    m_largeButton.Create("Крупные", WS_VISIBLE | WS_CHILD | WS_BORDER,
        CRect(400, 145, 490, 165), this, IDC_LISTVIEW_LARGE);
    m_listButton.Create("Список", WS_VISIBLE | WS_CHILD | WS_BORDER,
        CRect(400, 170, 490, 190), this, IDC_LISTVIEW_LIST);
    m_reportButton.Create("Таблица", WS_VISIBLE | WS_CHILD | WS_BORDER,
        CRect(400, 195, 490, 215), this, IDC_LISTVIEW_REPORT);
}
```

Сначала CreateListView() создает два списка изображений: один будет хранить малые пиктограммы, а второй — большие. В данном случае каждый список изображений хранит только по одной пиктограмме. В других приложениях вам могут понадобиться разные пиктограммы, например, для изображения папок, текстовых файлов и т.д. и еще список малых пиктограмм того же назначения.

Справившись со списками изображений, CreateListView() приступает к созданию просмотрового окна списка, вызывая, как обычно, функцию Create() этого класса:

```
// Создание просмотрового  окна списка.
  m_listView.Create(WS_VISIBLE | WS_CHILD | WS_BORDER |
    LVS_REPORT | LVS_NOSORTHEADER | LVS_EDITLABELS,
    CRect(160, 120, 394, 220), this, IDC_LISTVIEW);
```

В классе CListCtrl, объектом которого является m_listView, определены специальные константы стилей, которые можно использовать при создании экземпляра класса. В табл. 10.7 перечислены эти константы и даны пояснения относительно их назначения.

Таблица 10.7. Константы стиля для просмотрового окна списка

Стиль	Назначение
LVS_ALIGNLEFT	Размещает элементы списка вдоль левой границы окна при выводе в формате больших и малых пиктограмм
LVS_ALIGNTOP	Размещает элементы списка вдоль верхней границы окна при выводе в формате больших и малых пиктограмм
LVS_AUTOARRANGE	Автоматически размещает элементы списка в окне при выводе в формате больших и малых пиктограмм
LVS_EDITLABELS	Разрешает пользователю редактировать имя элемента
LVS_ICON	Устанавливает вывод в формате больших пиктограмм
LVS_LIST	Устанавливает вывод в формате списка
LVS_NOCOLUMNHEADER	Не показывает заголовки колонок при выводе в формате таблицы
LVS_NOITEMDATA	Сохраняет только состояние каждого элемента
LVS_NOLABELWRAP	Запрещает многострочные нвдписи на элементах
LVS_NOSCROLL	Запрещает прокрутку
LVS_NOSORTHEADER	Указывает, что заголовок колонки не может быть использован в качестве кнопки сортировки
LVS_OWNERDRAWFIXED	Разрешает использовать самостоятельно разработанную функцию отображения записей при выводе в формате таблицы
LVS_REPORT	Устанавливает вывод в формате таблицы
LVS_SHAREIMAGELISTS	Блокирует уничтожение элементом управления списка изображений после того, как элемент управления теряет необходимость в нем. Это позволяет использовать список изображений и другим элементам управления
LVS_SINGLESEL	Запрещает множественный выбор элементов списка
LVS_SMALLICON	Устанавливает вывод в формате малых пиктограмм
LVS_SORTASCENDING	Сортирует элементы списка в порядке возрастания
LVS_SORTDESCENDING	Сортирует элементы списка в порядке убывания

Третья задача CreateListView() — связать элемент управления с его списком изображений:

```
m_listView.SetImageList(&m_smallImageList, LVSIL_SMALL);
m_listView.SetImageList(&m_largeImageList, LVSIL_NORMAL);
```

Это выполняется двумя вызовами функции SetImageList(), которая имеет два аргумента — указатель списка изображений и флаг, задающий способ использования списка. Для этого флага определены три константы — LVSIL_SMALL (указывает, что список содержит малые пиктограммы), LVSIL_NORMAL (большие пиктограммы) и LVSIL_STATE (вывод состояния). SetImageList() возвращает указатель ранее установленного списка изображений, если таковой существовал.

Формирование колонок просмотрового окна списка

Четвертая задача CreateListView() — формирование колонок просмотрового окна списка для вывода в формате таблицы. Понадобится построить одну главную колонку с именами элементов списка и по одной колонке на каждый атрибут элемента. Например, в просмотровом окне списка Explorer главная (первая) колонка содержит имена файлов и папок. Каждая дополнительная колонка содержит какие-то атрибуты элементов — размер файла, тип, дату обновления и т.д. Для того чтобы сформировать колонку, нужно, во-первых, объявить структуру LV_COLUMN[1]. Эта структура используется для обмена информацией с системой. После того как колонка будет подключена к элементу управления с помощью функции InsertColumn(), можно будет использовать эту же структуру для формирования и подключения другой колонки. Структура LV_COLUMN представлена в листинге 10.7.

Листинг 10.7. Структура LV_COLUMN, объявленная в MFC

```
typedef struct _LV_COLUMN
{
   UINT mask;      // Флаги индикаторов действующих полей структуры.
   int fmt;      // Выравнивание колонки.
   int cx;      // Ширина колонки.
   LPSTR pszText;   // Адрес строкового буфера.
   int cchTextMax;   // Размер строкового буфера.
   int iSubItem;   // Индекс атрибута для этой колонки.
} LV_COLUMN;
```

Переменная-член структуры mask передает в систему информацию о том, какие другие члены структуры являются действительными в данном экземпляре, а какие могут быть проигиорированы. Выбор членов осуществляется с помощью следующих флагов.

- LVCF_FMT (действителен член fmt)
- LVCF_SUBITEM (действителен член iSubItem)
- LVCF_TEXT (действителен член pszText)
- LVCF_WIDTH (действителен член cx)

Член fmt задает способ выравнивания колонки и может принимать одно из следующих значений — LVCFMT_CENTER (центрировать), LVCFMT_LEFT (выравнивать по левому краю), LVCFMT_RIGHT (выравнивать по правому краю). Выравнивание относится к размещению текста заголовка и каждого элемента в колонке.

На заметку

Первая колонка, которая содержит главные атрибуты элементов, всегда выравнивается по левому краю. Другие колонки при выводе окна в формате таблицы выравниваются в соответствии с установленным программистом значением члена экземпляра структуры fmt.

---

[1] Фактически объявляется не структура, а ее экземпляр. Сама структура объявлена в MFC. — *Прим. ред.*

Поле cx определяет ширину колонки, а поле pszText является адресом строкового буфера. Когда вы используете такую структуру для формирования колонки (или когда она используется для считывания информации о состоянии колонки), в строковом буфере по заданному в этом поле адресу должен находиться текст заголовка колонки (или в него будет записан этот текст при считывании). Член-переменная cchTextMax определяет размер строкового буфера. Действует этот параметр только при считывании информации о колонке.

В функции CreateListView() создается временный экземпляр структуры LV_COLUMN, который заполняется информацией о формате вывода элементов списка и включается в просмотровое окно списка в качестве колонки 0 — главной колонки. Затем процесс повторяется для остальных двух колонок. Указанные действия выполняются в приведенном ниже фрагменте кода.

```
// Формирование колонок.
LV_COLUMN lvColumn;
lvColumn.mask = LVCF_FMT | LVCF_WIDTH | LVCF_TEXT | LVCF_SUBITEM;
lvColumn.fmt = LVCFMT_CENTER;
lvColumn.cx = 75;

lvColumn.iSubItem = 0;
lvColumn.pszText = "Кол. 0";
m_listView.InsertColumn(0, &lvColumn);
lvColumn.iSubItem = 1;
lvColumn.pszText = "Кол. 1";
m_listView.InsertColumn(1, &lvColumn);
lvColumn.iSubItem = 2;
lvColumn.pszText = "Кол. 2";
m_listView.InsertColumn(1, &lvColumn);
```

Формирование элементов для просмотрового окна списка

Пятая задача функции CreateListView() — сформировать элементы, которые должны быть перечислены в колонках этого окна при выводе в формате таблицы. Формирование элементов не намного отличается от формирования колонок. Как и в предыдущем случае Visual C++ предоставляет в распоряжение разработчика соответствующую структуру, экземпляр которой должен быть заполнен и передан в функцию, которая уже самостоятельно сформирует соответствующий элемент. Эта структура называется LV_ITEM, а ее объявление проиллюстрировано в листинге 10.8.

Листинг 10.8. Структура LV_ITEM, объявленная в MFC

```
typedef struct _LV_ITEM
{
   UINT mask;      // Флаги индикаторов действующих полей структуры.
   int iItem;      // Индекс элемента.
   int iSubItem;   // Индекс множества атрибутов этого элемента.
   UINT state;   // Текущее состояние элемента.
   UINT stateMask;   // Маска состояния.
   LPSTR pszText;   // Адрес строкового буфера.
   int cchTextMax;   // Размер строкового буфера.
   int iImage;   // Индекс картинки для этого элемента.
   LPARAM lParam;   // Дополнительная информация длиной 32 бит.
}.LV_ITEM;
```

Переменная-член структуры mask указывает с помощью флагов, какие другие члены структуры являются действительными в данном экземпляре, а какие могут быть проигнорированы. В распоряжении разработчика имеется следующий набор флагов.

- LVIF_IMAGE (переменная-член iImage содержит данные)
- LVIF_PARAM (переменная-член iParam содержит данные)
- LVIF_STATE (переменная-член state содержит данные)
- LVIF_TEXT (переменная-член pszText содержит данные)

Член iItem задает индекс элемента, который при выводе в формате таблицы есть не что иное как номер строки таблицы (но это только в случае, если строки не отсортированы по какому-либо атрибуту). Индекс каждого элемента уникален. Член-переменная iSubItem является индексом атрибута, если экземпляр структуры содержит данные об отдельном атрибуте. Его можно представить себе как номер колонки, в которой этот атрибут будет выведен. Например, для главного атрибута (он всегда выводится в первой колонке) эта величина должна быть равна 0.

Члены state и stateMask хранят текущее состояние элемента и допустимые состояния элемента соответственно. Константы состояний, как они определены в MFC, перечислены ниже.

- LVIS_CUT (элемент выбран для операций *вырезать и вставить* (Cut-and-Paste))
- LVIS_DROPHILITED (элемент выделен для переноса (drag-and-drop))
- LVIS_FOCUSED (элемент получил фокус)
- LVIS_SELECTED (элемент выбран)

Член pszText является адресом строкового буфера. Когда вы используете такую структуру для формирования элемента, в строковом буфере хранится соответствующий текст. Если будет считываться информация об элементе, то в pszText должен быть адрес строкового буфера-приемника, а член-переменная cchTextMax определяет размер строкового буфера. Если при установке pszText задана константа LPSTR_TEXTCALLBACK, то элемент будет использовать механизм *обратного вызова* (callback). И наконец, член-переменная iImage есть индекс соответствующей пиктограммы в списке изображений больших и малых пиктограмм. Если при установке iImage задана константа I_IMAGECALLBACK, то элемент будет использовать механизм *обратного вызова*.

В функции CreateListView()создается временный экземпляр структуры LV_ITEM, который заполняется соответствующей информацией и включается в просмотровое окно списка в качестве элемента 0. Два вызова функции SetItemText добавляют информацию об атрибутах этого элемента, так что готовится текст для каждой колонки таблицы. Далее весь процесс повторяется для оставшихся двух элементов списка. Ниже представлен соответствующий фрагмент программного кода функции.

```
// Формирование элементов списка.
LV_ITEM lvItem;
lvItem.mask = LVIF_TEXT | LVIF_IMAGE | LVIF_STATE;
lvItem.state = 0;
lvItem.stateMask = 0;
lvItem.iImage = 0;

lvItem.iItem = 0;
lvItem.iSubItem = 0;
lvItem.pszText = "Элем. 0";
m_listView.InsertItem(&lvItem);
m_listView.SetItemText(0, 1, "Вх. 0.1");
m_listView.SetItemText(0, 2, "Вх. 0.2");
```

```
lvItem.iItem = 1;
lvItem.iSubItem = 0;
lvItem.pszText = "Элем. 1";
m_listView.InsertItem(&lvItem);
m_listView.SetItemText(1, 1, "Вх. 1.1");
m_listView.SetItemText(1, 2, "Вх. 1.2");

lvItem.iItem = 2;
lvItem.iSubItem = 0;
lvItem.pszText = "Элем. 2";
m_listView.InsertItem(&lvItem);
m_listView.SetItemText(2, 1, "Вх. 2.1");
m_listView.SetItemText(2, 2, "Вх. 2.2");
```

Теперь будет сформировано просмотровое окно списка с тремя колонками, а в списке будет три элемента. В реальных (не демонстрационных) приложениях этот процесс не так жестко запрограммирован (в смысле количества элементов и содержимого их атрибутов), а данные чаще всего готовятся самой программой.

Манипулирование просмотровым окном списка

Можно задать четыре формата вывода информации в просмотровом окне списка — малые и большие пиктограммы, простой список и реестр. В Explorer, например, для управления форматом используется специальная панель инструментов или команды меню View. Хотя возможности приложения Common и Explorer Win95 соотносятся примерно так же, как возможности небезызвестной Фимы Собак и малоизвестной, но любимой женщины миллионера Вандербильда, кое-какими средствами оно все-таки располагает. Это кое-что имеет вид четырех кнопок — Мелкие (Small), Крупные (Large), Список (List) и Таблица (Report). Щелкнув на любой из них, можно изменить формат представления элементов списка. Формирование этих кнопок есть шестая задача функции CreateListView().

```
// Формирование кнопок управления просмотром.
m_smallButton.Create("Мелкие", WS_VISIBLE | WS_CHILD | WS_BORDER,
   CRect(400, 120, 490, 140), this, IDC_LISTVIEW_SMALL);
m_largeButton.Create("Крупные", WS_VISIBLE | WS_CHILD | WS_BORDER,
   CRect(400, 145, 490, 165), this, IDC_LISTVIEW_LARGE);
m_listButton.Create("Список", WS_VISIBLE | WS_CHILD | WS_BORDER,
   CRect(400, 170, 490, 190), this, IDC_LISTVIEW_LIST);
m_reportButton.Create("Таблица", WS_VISIBLE | WS_CHILD | WS_BORDER,
   CRect(400, 195, 490, 215), this, IDC_LISTVIEW_REPORT);
```

Эти кнопки ассоциированы посредством введенных вручную элементов карты сообщений фрейма окна приложения с функциями обработки сообщений OnSmall(), OnLarge(), OnList() и OnReport().

Отредактируйте карту сообщений в файле заголовка CommonView.h и определите обработчики для каждой из этих кнопок.

```
// Сформированные функции карты сообщений.
protected:

   //{{AFX_MSG(CCommonView)
   afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
   afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
   afx_msg void OnDestroy();
   afx_msg void OnTimer(UINT nIDEvent);
   afx_msg void OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar);
   //}}AFX_MSG
   afx_msg void OnSmall();
   afx_msg void OnLarge ();
```

```
   afx_msg void OnList();
   afx_msg void OnReport();
   DECLARE_MESSAGE_MAP()
};
```

Отредактируйте карту сообщений и в файле реализации CommonView.cpp и свяжите сообщения с этими функциями.

```
BEGIN_MESSAGE_MAP( CCommonView, CScrollView)
   //{{AFX_MSG_MAP(CCommonView)
   ON_WM_CREATE()
   ON_WM_LBUTTONDOWN()
   ON_WM_DESTROY()
   ON_WM_TIMER()
   ON_WM_HSCROLL()
   //}}AFX_MSG_MAP
   ON_WM_COMMAND(IDC_LISTVIEW_SMALL, OnSmall)
   ON_WM_COMMAND(IDC_LISTVIEW_LARGE, OnLarge)
   ON_WM_COMMAND(IDC_LISTVIEW_LIST, OnList)
   ON_WM_COMMAND(IDC_LISTVIEW_REPORT, OnReport)
   //Стандартные команды вывода на печать
   ON_WM_COMMAND(ID_FILE_PRINT, CScrollView::DnFilePrint)
   ON_WM_COMMAND(ID_FILE_PRINT_DIRECT, CScrollView::OnFilePrint)
   ON_WM_COMMAND(ID_FILE_PRINT_PREVIEW, CScrollView::OnFilePrintPreview)
END_MESSAGE_MAP()
```

Теперь выберите View⇨Resource Symbols, щелкните на кнопке New и добавьте новые значения для каждого идентификатора в приведенных выше фрагментах программного кода.

- IDC_LISTVIEW
- IDC_LISTVIEW_SMALL
- IDC_LISTVIEW_LARGE
- IDC_LISTVIEW_LIST
- IDC_LISTVIEW_REPORT

Каждый из четырех обработчиков вызывает функцию SetWindowLong(), которая устанавливает атрибуты окна. Ее аргументами являются дескриптор окна, флаг, специфицирующий изменяемый атрибут, и новое значение атрибута. Например, передача константы GWL_STYLE в качестве второго аргумента указывает, что будет изменен стиль окна, причем новый стиль определяется третьим аргументом функции. Изменение стиля окна представления списка (например, на LVS_SMALLICON) задает новый формат вывода списка. Приняв все сказанное во внимание, добавьте в конец файла CommonView.cpp коды этих четырех функций.

```
void CCommonView::OnSmall()
{
  SetWindowLong(m_listView.m_hWnd, GWL_STYLE,
    WS_VISIBLE | WS_CHILD | WS_BORDER |
    LVS_SMALLICON | LVS_EDITLABELS);
}
void CCommonView::OnLarge ()
{
  SetWindowLong(m_listView.m_hWnd, GWL_STYLE,
    WS_VISIBLE | WS_CHILD | WS_BORDER |
    LVS_ICON | LVS_EDITLABELS);
}
void CCommonView::OnList()
{
  SetWindowLong(m_listView.m_hWnd, GWL_STYLE,
    WS_VISIBLE | WS_CHILD | WS_BORDER |
    LVS_LIST | LVS_EDITLABELS);
}
```

```
void CCommonView::OnReport()
{
  SetWindowLong(m_listView.m_hWnd, GWL_STYLE,
    WS_VISIBLE | WS_CHILD | WS_BORDER |
    LVS_REPORT | LVS_EDITLABELS);
}
```

Кроме возможности изменения формата вывода, в распоряжении разработчика имеется множество средств манипулирования этим элементом управления. Когда пользователь что-либо выполняет в окне, Windows посылает сообщение WM_NOTIFY родительскому окну. В ответной реакции на это сообщение можно запрограммировать любые возможности манипулирования списком, какие вы сочтете нужным. Чаще всего просмотровое окно списка передает следующие уведомления в сообщении WM_NOTIFY.

- LVN_COLUMNCLICK (пользователь щелкнул на заголовке колонки)
- LVN_BEGINLABELEDIT (пользователь собирается редактировать надпись элемента)
- LVN_ENDLABELEDIT (пользователь закончил редактировать надпись элемента)

Как организовать, например, редактирование первой колонки в списке? Первым делом, нужно перегрузить виртуальную функцию OnNotify(), которая унаследована классом CCommonView от CScrollView. Щелкните правой кнопкой мыши на строке CCommonView в окне ClassView и выберите в контекстном меню Add Virtual Function. В списке слева выберите OnNotify() и щелкните на кнопке Add and Edit. Затем добавьте следующие операторы в начало текста программы этой функции, заменив ими комментарий TODO:

```
LV_DISPINFO* lv_dispInfo = (LV_DISPINFO*) lParam;

  if (lv_dispInfo->hdr.code == LVN_BEGINLABELEDIT)
  {
    CEdit* pEdit = m_listView.GetEditControl();
  }
  else if (lv_dispInfo->hdr.code == LVN_ENDLABELEDIT)
  {
    if ((lv_dispInfo->item.pszText != NULL) &&
      (lv_dispInfo->item.iItem != -1))
    {
      m_listView.SetItemText(lv_dispInfo->item.iItem,
        0, lv_dispInfo->item.pszText);
    }
  }
```

Функция OnNotify() получает три аргумента — параметры WPARAM и LPARAM сообщения и указатель кода результата. В данном случае сообщение WM_NOTIFY выбирается из просмотрового окна списка и параметр WPARAM является идентификатором самого элемента управления. Если сообщение WM_NOTIFY сопровождается уведомлением LVN_BEGINLABELEDIT или LVN_ENDLABELEDIT, то LPARAM есть указатель на структуру LV_DISPINFO, которая, в свою очередь, содержит структуры NMHDR и LV_ITEM. Информацию, содержащуюся в этих структурах, можно использовать для обработки элемента, который пользователь собрался редактировать.

Если будет получено уведомление LVN_BEGINLABELEDIT, программа сможет выполнить необходимые действия по инициализации, предшествующие редактированию. Для этого вызывается функция GetEditControl(), а затем обрабатывается возвращенный ею указатель. В нашем простом приложении только продемонстрировано, как получить этот указатель.

Во время редактирования надписи нужно все время отслеживать появление сообщения с уточнением LVN_ENDLABELEDIT. Оно означает, что пользователь закончил редактирование и дал об этом знать, либо завершив ввод, либо отказавшись от изменения надписи. Если пользователь отказался, член item.pszText структуры LV_DISPINFO примет значение NULL или член item.iItem будет равен –1. В этом случае не остается ничего другого, как проигнорировать

уведомление. Если, однако, пользователь успешно закончил редактирование, программа должна скопировать введенный текст в поле текста элемента списка. Это выполняет функция `OnNotify()`, вызывая, в свою очередь, функцию `SetItemText()`. Метод `SetItemText()` класса `CListCtrl` имеет три аргумента — индекс элемента, индекс атрибута элемента и новый текст.

Теперь можно вновь оттранслировать приложение и протестировать его. Щелкните на каждой из четырех кнопок и посмотрите, как будет изменяться формат представления списка. Попробуйте также отредактировать одну из надписей в первой колонке списка.

На рис. 10.1 вы уже могли увидеть окно списка в формате таблицы. На рис. 10.6 это же окно показано в формате малых пиктограмм, а на рис. 10.7 — в формате больших пиктограмм.

С просмотровым окном списка можно сделать много довольно интересных операций, и мы рекомендуем вам не пожалеть времени на эксперименты с ним. Освоив методику программирования элемента управления этого типа, вы сможете значительно ускорить разработку собственных приложений, имеющих дело с массивами разного рода информационных структур.

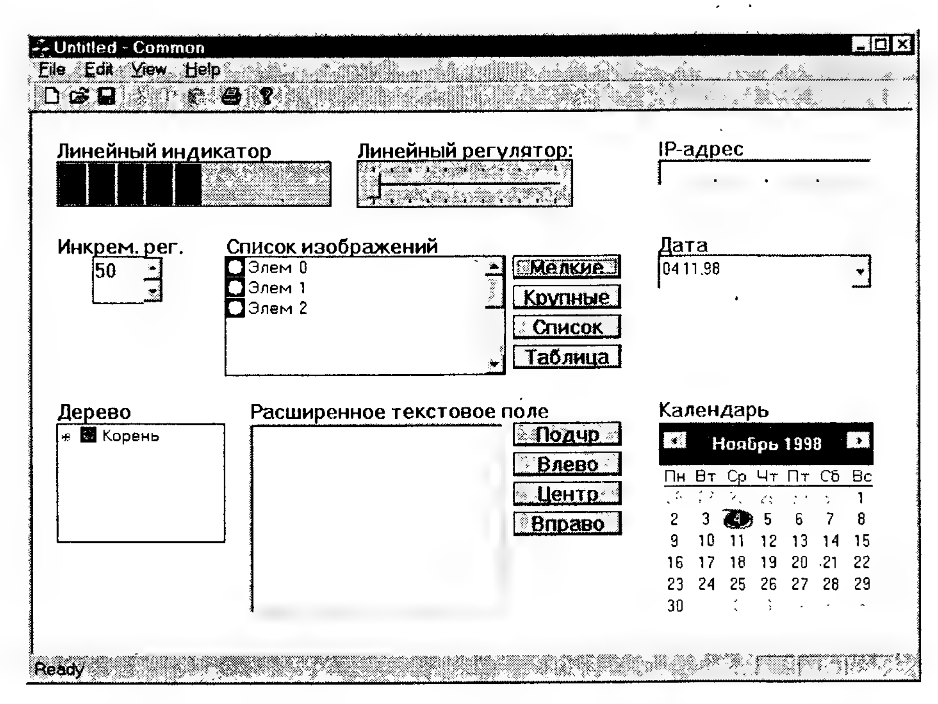


Рис. 10.6. Просмотровое окно списка приложения Common, настроенное на формат малых пиктограмм

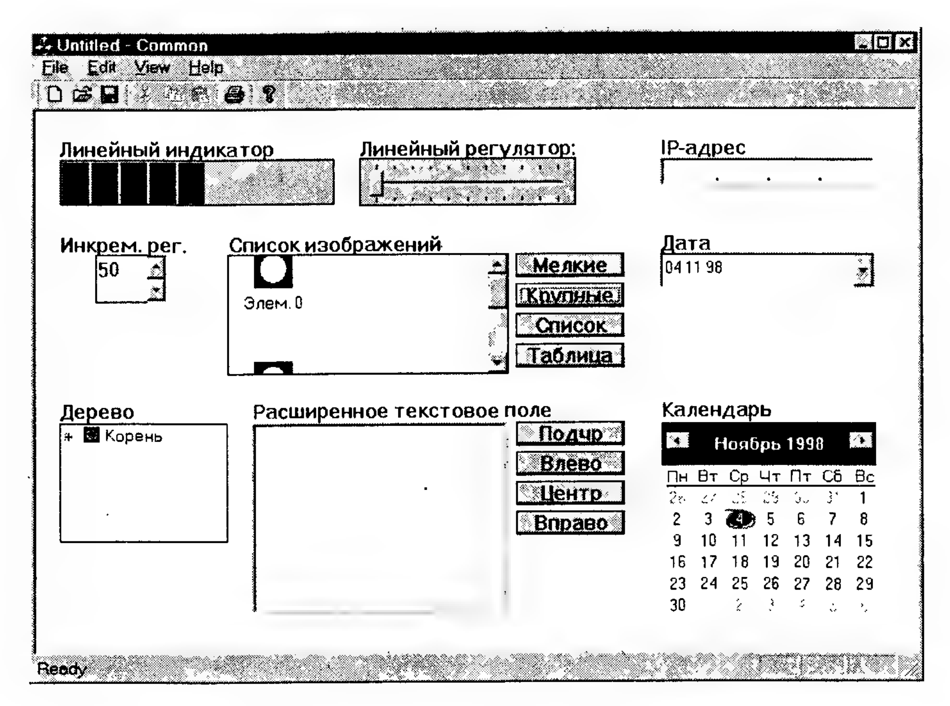


Рис. 10.7. Просмотровое окно списка приложения Common, настроенное на формат больших пиктограмм

Просмотровое окно дерева

В предыдущем разделе вы научились использовать просмотровое окно списка для организации отображения множества объектов в окне и манипулирования ими. Элемент управления этого типа позволяет организовать отображение объектов либо в виде множества пиктограмм, покрывающих поле окна, либо в виде многоколонной таблицы. Однако существуют такие области применения, где объекты лучше всего представлять пользователю в виде иерархической структуры. В этом случае можно наглядно продемонстрировать связь элементов между собой. Наиболее распространенным примером такого представления объектов является всем известное дерево файлов и папок.

Как и в случае с другими полезными элементами управления, Windows 95 включает просмотровое окно дерева в состав своих стандартных средств. В MFC для операций с элементами управления этого типа существует класс `CTreeCtrl`. С помощью такого просмотрового окна можно отображать данные о самых разнообразных объектах, связанных между собой некоторыми отношениями иерархической подчиненности.

Пример просмотрового окна дерева можно найти в том же Explorer Windows 95 (рис. 10.8). В левой части окна программы сформировано просмотровое окно дерева, в котором показана иерархия объектов хранения информации в компьютере. (В правой части — уже известное вам просмотровое окно списка.) Как видно, в иерархию включены и устройства хранения информации (фактически — логические тома), и папки, и отдельные файлы. Древовидная структура изображения отражает иерархию подчиненности объектов, причем пользователь может по своему желанию “разворачивать” (открывать) или “сворачивать” (закрывать) отдельные ветви и уровни иерархии.

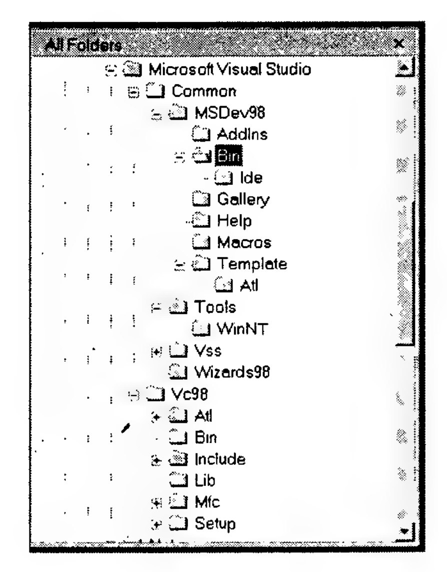


Рис. 10.8. Просмотровое окно дерева отображает иерархию связей между объектами

Формирование просмотрового окна дерева

В приложении Common для создания элемента управления типа просмотровое окно дерева необходимо добавить в класс представления локальный метод CreateTreeView(), который, в свою очередь, вызывается методом OnCreate(). Текст CCommonView::CreateTreeView() представлен в листинге 10.9. Последовательность действий, выполняемых CreateTreeView(), следующая.

1. Создать список изображений.
2. Создать сам элемент управления типа просмотровое окно дерева.
3. Связать список изображений с просмотровым окном дерева.
4. Сформировать корневой элемент дерева.
5. Сформировать элементы-потомки.

Листинг 10.9. Файл CommonView.cpp — функция CCommonView::CreateTreeView()

```
void CCommonView::CreateTreeView()
{
   // Создать список изображений.
  m_treeImageList.Create(13, 13, FALSE, 3, 0);
  HICON hIcon = ::LoadIcon(AfxGetResourceHandle(),
    MAKEINTRESOURCE(IDI_ICON3));
  m_treeImageList.Add(hIcon);
  hIcon = ::LoadIcon(AfxGetResourceHandle(),
    MAKEINTRESOURCE(IDI_ICON4));
  m_treeImageList.Add(hIcon);
  hIcon = ::LoadIcon(AfxGetResourceHandle(),
    MAKEINTRESOURCE(IDI_ICON5));
  m_treeImageList.Add(hIcon);
```

```
    // Create the Tree View control.
    // Создать просмотровое окно дерева.
    m_treeView.Create(WS_VISIBLE | WS_CHILD | WS_BORDER |
        TVS_HASLINES | TVS_LINESATROOT | TVS_HASBUTTONS |
        TVS_EDITLABELS, CRect(20, 260, 160, 360), this,
        IDC_TREEVIEW);
    m_treeView.SetImageList(&m_treeImageList, TVSIL_NORMAL);

    // Create the root item.
    // Сформировать корневой элемент.
    TVITEM tvItem;
    tvItem.mask =
        TVIF_TEXT | TVIF_IMAGE | TVIF_SELECTEDIMAGE;
    tvItem.pszText = "Корень";
    tvItem.cchTextMax = 4;
    tvItem.iImage = 0;
    tvItem.iSelectedImage = 0;
    TVINSERTSTRUCT tvInsert;
    tvInsert.hParent = TVI_ROOT;
    tvInsert.hInsertAfter = TVI_FIRST;
    tvInsert.item = tvItem;
    HTREEITEM hRoot = m_treeView.InsertItem(&tvInsert);

    // Create the first child item.
    // Сформировать первый элемент-потомок.
    tvItem.pszText = "Дочерний элемент 1";
    tvItem.cchTextMax = 12;
    tvItem.iImage = 1;
    tvItem.iSelectedImage = 1;
    tvInsert.hParent = hRoot;
    tvInsert.hInsertAfter = TVI_FIRST;
    tvInsert.item = tvItem;
    HTREEITEM hChildItem = m_treeView.InsertItem(&tvInsert);

    // Create a child of the first child item.
    // Сформировать потомок первого элемента-потомка.
    tvItem.pszText = "Дочерний элемент 2";
    tvItem.cchTextMax = 12;
    tvItem.iImage = 2;
    tvItem.iSelectedImage = 2;
    tvInsert.hParent = hChildItem;
    tvInsert.hInsertAfter = TVI_FIRST;
    tvInsert.item = tvItem;
    m_treeView.InsertItem(&tvInsert);

    // Create another child of the root item.
    // Сформировать другой потомок корневого элемента.
    tvItem.pszText = "Дочерний элемент 3";
    tvItem.cchTextMax = 12;
    tvItem.iImage = 1;
    tvItem.iSelectedImage = 1;
    tvInsert.hParent = hRoot;
    tvInsert.hInsertAfter = TVI_LAST;
    tvInsert.item = tvItem;
    m_treeView.InsertItem(&tvInsert);
}
```

Создание списка изображений, самого элемента управления типа просмотровое окно дерева и организация связи между списком изображений и просмотровым окном дерева очень похожи на аналогичные стадии формирования просмотрового окна списка. Собственно создание элемента управления выполняется операторами.

```
// Создать просмотровое окно дерева.
  m_treeView.Create(WS_VISIBLE | WS_CHILD | WS_BORDER |
    TVS_HASLINES | TVS_LINESATROOT | TVS_HASBUTTONS |
    TVS_EDITLABELS, CRect(20, 260, 160, 360), this,
    IDC_TREEVIEW);
  m_treeView.SetImageList(&m_treeImageList, TVSIL_NORMAL);
```

(Не забудьте определить идентификатор ресурса IDC_TREEVIEW.) Класс CTreeCtrl, экземпляром которого в программе является m_treeView, располагает набором собственных констант стилей, специфических именно для отображения деревьев. Они перечислены в табл. 10.8.

Таблица 10.8. Константы стиля для просмотрового окна дерева

Стиль	Назначение
TVS_DISABLEDRAGDROP	Запрещает операции "перетащить и опустить" (drag-and-drop)
TVS_EDITLABELS	Разрешает пользователю редактировать имя элемента
TVS_HASBUTTONS	Присваивает каждому узлу-родителю кнопку сворачивания-разворачивания
TVS_HASLINES	Соединяет потомки линиями на экране с их родительскими записями
TVS_LINESATROOT	Соединяет записи первого уровня линиями с вершиной дерева
TVS_SHOWSELALWAYS	Оставляет выбранный элемент в этом же состоянии и после того, как окно теряет фокус ввода

Формирование элементов просмотрового окна дерева

Принципиально технологии создания элементов для просмотровых окон дерева и списка не различаются. Эта стадия во многом напоминает аналогичный процесс для просмотрового окна списка.

Как и в предыдущем случае, Visual C++ располагает структурой, экземпляр которой должен быть создан, заполнен данными и передан в функцию, которая и сформирует элемент. Эта структура называется TVITEM; в листинге 10.10 представлен текст ее объявления.

Листинг 10.10. Структура TVITEM, объявленная в MFC

```
typedef struct _TVITEM
{
   UINT      mask;
   HTREEITEM hItem;
   UINT      state;
   UINT      stateMask;
   LPSTR     pszText;
   int       cchTextMax;
   int       iImage;
   int       iSelectedImage;
   int       cChildren;
   LPARAM    lParam;
} TVITEM;
```

Переменная-член структуры mask указывает посредством флагов, какие другие члены структуры являются действительными в данном экземпляре, а какие могут быть проигнорированы. В распоряжении разработчика имеется следующий набор флагов.

- TVIF_CHILDREN (член cChildren содержит данные)
- TVIF_HANDLE (член hItem содержит данные)
- TVIF_IMAGE (член iImage содержит данные)
- TVIF_PARAM (член iParam содержит данные)
- TVIF_SELECTEDIMAGE (член iSelectedImage содержит данные)
- TVIF_STATE (члены state и stateMask содержат данные)
- TVIF_TEXT (члены pszText и cchTextMax содержат данные)

Член hItem задает дескриптор элемента, а члены state и stateMask хранят текущее состояние элемента и допустимые состояния элемента соответственно. Значения этих членов задается как комбинация констант состояний, определенных в MFC: TVIS_BOLD, TVIS_CUT, TVIS_DROPHILITED, TVIS_EXPANDED, TVIS_EXPANDEDONCE, TVIS_FOCUSED, TVIS_OVERLAYMASK, TVIS_SELECTED, TVIS_STATEIMAGEMASK и TVIS_USERMASK.

Член pszText является адресом строкового буфера. Когда вы используете такую структуру для формирования элемента, в строковом буфере хранится соответствующий текст. Если будет считываться информация об элементе, то в pszText должен быть адрес строкового буфера-приемника, а член-переменная cchTextMax определяет размер строкового буфера. Если в качестве значения pszText задана константа LPSTR_TEXTCALLBACK, то элемент будет использовать механизм *обратного вызова* (callback). Член-переменная iImage есть индекс соответствующей пиктограммы в списке изображений. Если при установке iImage задана константа I_IMAGECALLBACK, то элемент будет использовать механизм *обратного вызова*.

Член iSelectedImage есть индекс пиктограммы, используемой для представления элемента, когда он выделен соответствующей пиктограммой в списке изображений больших и малых пиктограмм. Как и в случае с iImage, если при установке этого члена задана константа I_IMAGECALLBACK, элемент будет использовать механизм *обратного вызова*. И наконец, член cChildren указывает, имеет ли данный элемент связанные с ним элементы-потомки.

Кроме структуры TVITEM, нужно будет инициализировать и экземпляр структуры TVINSERTSTRUCT, которая хранит информацию о том, как вставить новую структуру в просмотровое окно дерева. Текст объявления структуры TVINSERTSTRUCT представлен в листинге 10.11.

Листинг 10.11. Структура TVINSERTSTRUCT, объявленная в MFC

```
typedef struct tagTVINSERTSTRUCT {
   HTREEITEM hParent;
   HTREEITEM hInsertAfter;
#if (_WIN32_IE >= 0x0400)
   union
   {
      TVITEMEX itemex;
      TVITEM item;
   }
#else
   TVITEM item;
#endif
} TVINSERTSTRUCT, FAR *LPTVINSERTSTRUCT;
```

В этой структуре hParent есть дескриптор элемента-родителя для данного элемента. Значение NULL или TVI_ROOT для этого члена — признак того, что данный элемент является корневым в древовидной структуре. Член hInsertAfter указывает дескриптор элемента, *перед* которым должен быть вставлен текущий. Его значение может также иметь вид одного из флагов: TVI_FIRST (начало списка), TVI_LAST (конец списка) или TVI_SORT (в алфавитном по-

рядке). Член item представляет собой экземпляр структуры TVITEM, в котором содержится информация о самом вставляемом элементе.

В приложении Common функция CreateTreeView(), в первую очередь, создает экземпляр структуры TVITEM для корневого элемента (первого элемента дерева).

```
// Сформировать корневой элемент.
  TVITEM tvItem;
  tvItem.mask =
    TVIF_TEXT | TVIF_IMAGE | TVIF_SELECTEDIMAGE;
  tvItem.pszText = "Корень";
  tvItem.cchTextMax = 4;
  tvItem.iImage = 0;
  tvItem.iSelectedImage = 0;
  TVINSERTSTRUCT tvInsert;
  tvInsert.hParent = TVI_ROOT;
  tvInsert.hInsertAfter = TVI_FIRST;
  tvInsert.item = tvItem;
  HTREEITEM hRoot = m_treeView.InsertItem(&tvInsert);
```

Метод InsertItem класса CTreeCtrl берет на себя все заботы о включении элемента в древовидную структуру. Единственным ее аргументом является адрес соответствующего экземпляра структуры TVINSERTSTRUCT.

А затем функция CreateTreeView() включает в дерево все остальные элементы.

```
// Сформировать первый элемент-потомок.
  tvItem.pszText = "Дочерний элемент 1";
  tvItem.cchTextMax = 12;
  tvItem.iImage = 1;
  tvItem.iSelectedImage = 1;
  tvInsert.hParent = hRoot;
  tvInsert.hInsertAfter = TVI_FIRST;
  tvInsert.item = tvItem;
  HTREEITEM hChildItem = m_treeView.InsertItem(&tvInsert);

  // Сформировать потомок первого элемента-потомка.
    tvItem.pszText = "Дочерний элемент 2";
  tvItem.cchTextMax = 12;
  tvItem.iImage = 2;
  tvItem.iSelectedImage = 2;
  tvInsert.hParent = hChildItem;
  tvInsert.hInsertAfter = TVI_FIRST;
  tvInsert.item = tvItem;
  m_treeView.InsertItem(&tvInsert);

  // Сформировать другой потомок корневого элемента.
  tvItem.pszText = "Дочерний элемент 3";
  tvItem.cchTextMax = 12;
  tvItem.iImage = 1;
  tvItem.iSelectedImage = 1;
  tvInsert.hParent = hRoot;
  tvInsert.hInsertAfter = TVI_LAST;
  tvInsert.item = tvItem;
  m_treeView.InsertItem(&tvInsert);
```

Манипулирование просмотровым окном дерева

Просмотровое окно дерева в приложении Common позволяет точно так, как и просмотровое окно списка, редактировать надписи на элементах. Как и в предыдущем случае, это осуществляется в ответ на сообщение WM_NOTIFY, которое запускает функцию OnNotify().

Функция OnNotify() обрабатывает уведомления просмотрового окна дерева так же, как и уведомления просмотрового окна списка. Единственная разница — в именах используемых структур. Добавьте в текст функции OnNotify() перед оператором return следующий код.

```
TV_DISPINFO* tv_dispInfo = (TV_DISPINFO*) lParam;

  if (tv_dispInfo->hdr.code == TVN_BEGINLABELEDIT)
  {
    CEdit* pEdit = m_treeView.GetEditControl();
  }
  else if (tv_dispInfo->hdr.code == TVN_ENDLABELEDIT)
  {
    if (tv_dispInfo->item.pszText != NULL)
    {
      m_treeView.SetItemText(tv_dispInfo->item.hItem,
        tv_dispInfo->item.pszText);
    }
  }
```

Просмотровое окно дерева передает вместе с сообщением WM_NOTIFY много других уведомлений, в число которых входят TVN_BEGINDRAG, TVN_BEGINLABELEDIT, TVN_BEGINRDRAG, TVN_DELETEITEM, TVN_ENDLABELEDIT, TVN_GETDISPINFO, TVN_ITEMEXPANDED, TVN_ITEMEXPANDING, TVN_KEYDOWN, TVN_SELCHANGED, TVN_SELCHANGING и TVN_SETDISPINFO. Подробную информацию об обработке этих уведомлений можно найти в оперативной справке Visual C++.

Теперь можно снова оттранслировать приложение с новым элементом управления и протестировать его.

Расширенное текстовое поле

Если бы в свое время можно было всю энергию, потраченную на программирование операций редактирования текстов, использовать для решения других, менее занудных, но более полезных задач, компьютерная наука сейчас ушла бы на десяток лет вперед. Хотя это утверждение и выглядит несколько экстравагантным, никто не будет отрицать такого очевидного факта, что редактирование текстов — это область программирования, в которой один и тот же велосипед изобретался не один десяток раз. И уже в те времена хрустальной мечтой программистов было иметь некий стандартный инструмент для редактирования текстов, который можно было бы включать в любое приложение и при желании добавлять в него какие-нибудь свои специфические функции.

Класс CRichEditCtrl из библиотеки MFC предоставляет в распоряжение программиста такой элемент управления, который можно использовать для включения в приложение редактора текстов с достаточно развитыми возможностями. Элемент управления типа расширенное текстовое поле располагает такими функциями, как управление шрифтами, стилями абзацев, цветом текста и другими, которые стали стандартными для более или менее порядочного текстового редактора. Фактически расширенное текстовое поле (rich edit control) (получившее свое наименование от Rich Text Format — формата, в котором обрабатывается текст в этом элементе управления) — представляет собой идеальный инструмент, возможности которого можно наращивать и адаптировать к любой конкретной задаче, связанной с обработкой текстов в приложении. В частности, он позволяет пользователю выполнять следующие операции.

- Вводить текст с клавиатуры.
- Редактировать текст, используя распространенные технологии "вырезать и вставить" (cut-and-paste) и "перетащить и опустить" (drag-and-drop).
- Устанавливать атрибуты фрагментов текста, такие как шрифт, размер символов и цвет.
- Применять к фрагментам текста такие опции, как подчеркивание, выделение полужирным шрифтом или курсивом, перечеркивание, вывод в виде надстрочного или подстрочного текста.
- Форматировать текст с различными вариантами выравнивания и оформления перечней.
- Блокировать дальнейшее редактирование текста.
- Обмениваться текстом с файловой системой — записывать и считывать текстовые файлы.

Из всего сказанного следует, что расширенное текстовое поле есть довольно мощный инструмент работы с текстами. Фактически это готовый встраиваемый текстовый редактор, который можно включить в разрабатываемое приложение и сразу же использовать. Но, как это бывает с любым многофункциональным инструментом, придется потратить немало времени на то, чтобы освоить все его богатые возможности. В этом разделе мы вкратце познакомим вас с созданием этих элементов управления и манипулированием ими.

Создание расширенного текстового поля

В приложении Common расширенное текстовое поле создается методом CreateRichEdit() класса CCommonView, текст которого представлен в листинге 10.12. Этот метод в свою очередь вызывается методом OnCreate() этого же класса.

Листинг 10.12. Файл CommonView.cpp — функция CCommonView::CreateRichEdit()

```
void CCommonView::CreateRichEdit()
{
    m_richEdit.Create(WS_CHILD | WS_VISIBLE | WS_BORDER |
        ES_AUTOVSCROLL | ES_MULTILINE,
        CRect(180, 260, 393, 420), this, IDC_RICHEDIT);
    m_boldButton.Create("Подчр", WS_VISIBLE | WS_CHILD | WS_BORDER,
        CRect(400, 260, 490, 280), this, IDC_RICHEDIT_ULINE);
    m_leftButton.Create("Влево", WS_VISIBLE | WS_CHILD | WS_BORDER,
        CRect(400, 285, 490, 305), this, IDC_RICHEDIT_LEFT);
    m_centerButton.Create("Центр", WS_VISIBLE | WS_CHILD | WS_BORDER,
        CRect(400, 310, 490, 330), this, IDC_RICHEDIT_CENTER);
    m_rightButton.Create("Вправо", WS_VISIBLE | WS_CHILD | WS_BORDER,
        CRect(400, 335, 490, 355), this, IDC_RICHEDIT_RIGHT);
}
```

Как обычно, все начинается с вызова функции Create(), члена класса, соответствующего создаваемому элементу управления (в данном случае — CRichEditCtrl). Флаги стиля задаются как комбинация символов констант стиля, в набор которых входят как константы, применимые для всех окон, так и специально созданные для расширенного текстового поля. В табл. 10.9 представлены эти специальные константы стиля.

Таблица 10.9. Константы стиля для расширенного текстового поля

Стиль	Назначение
ES_AUTOHSCROLL	Разрешает автоматическую горизонтальную прокрутку текста
ES_AUTOVSCROLL	Разрешает автоматическую вертикальную прокрутку текста
ES_CENTER	Центрирует текст
ES_LEFT	Выравнивает текст по левой кромке поля
ES_LOWERCASE	Выводит весь текст строчными литерами
ES_MULTILINE	Разрешает многострочный вывод текста
ES_NOHIDESEL	Оставляет выделение выбранного текста после того, как элемент потеряет фокус ввода
ES_OEMCONVERT	Преобразует символы из формата ANSI в OEM и наоборот
ES_PASSWORD	Отображает все символы в виде *звездочек* (asterisks)
ES_READONLY	Запрещает редактирование текста
ES_RIGHT	Выравнивает текст по правой границе поля
ES_UPPERCASE	Выводит весь текст прописными буквами
ES_WANTRETURN	Вставляет в текст символы перевода каретки при нажатии клавиши <Enter> во время ввода

Инициализация расширенного текстового поля

Созданный экземпляр класса CRichEditCtrl необходимо некоторым образом инициализировать. (В приложении Common не выполняется никаких специальных действий по инициализации этого элемента управления, но, тем не менее, он прекрасно работает сразу же после создания.) В классе CRichEditCtrl имеется большой набор функций-членов, которые позволяют по-своему инициализировать элемент управления и манипулировать им. Перечень этих функций приведен в табл. 10.10, а детали их применения — в электронной справке Visual C++.

Таблица 10.10. Функции-члены класса CRichEditCtrl

Функция	Назначение
CanPaste()	Определяет взаимоотношения элемента управления и системного буфера Clipboard — можно или нельзя вставлять его содержимое в элемент управления
CanUndo()	Определяет, можно ли будет выполнить откат, аннулировав последнюю операцию редактирования
Clear()	Удаляет выбранный текст
Copy()	Копирует выбранный текст в системный буфер Clipboard
Create()	Создает новый элемент управления
Cut()	Вырезает и копирует выбранный текст в системный буфер Clipboard
DisplayBand()	Отображает часть текста из буфера элемента управления
EmptyUndoBuffer()	Сбрасывает флаг отката
FindText()	Находит заданный текст
FormatRange()	Форматирует текст для заданного выходного устройства
GetCharPos()	Считывает позицию заданного символа
GetDefaultCharFormat()	Считывает формат символа, заданный по умолчанию
GetEventMask()	Считывает маску событий для элемента управления
GetFirstVisibleLine()	Считывает индекс первой видимой строки

Функция	Назначение
GetRichEditOle()	Считывает указатель интерфейса IRichEditOle для элемента управления
GetLimitText()	Считывает значение параметра — максимального количества символов, которые можно ввести в элемент управления
GetLine()	Считывает указанную строку текста
GetLineCount()	Считывает значение параметра — фактического количества текстовых строк в элементе управления
GetModify()	Определяет, производилось ли изменение текста после последней операции сохранения
GetParaFormat()	Считывает формат абзаца для выделенного текста
GetRect()	Считывает прямоугольник форматирования, установленный для элемента
GetSel()	Считывает положение текущего выбранного текста
GetSelectionCharFormat()	Считывает формат символа, заданный для выделенного текста
GetSelectionType()	Считывает тип содержимого выделенного текста
GetSelText()	Считывает текущий выбранный текст
GetTextLength()	Считывает текущую длину текста в элементе управления
HideSelection()	Скрывает или восстанавливает изображение выделенного текста
LimitText()	Устанавливает значение параметра — максимального количества символов, которые можно ввести в элемент управления
LineFromChar()	Считывает номер строки, в которой находится данный символ
LineIndex()	Считывает индекс символа в данной строке
LineLength()	Считывает длину заданной строки
LineScroll()	Продлевает текст на заданное количество строк и символов
Paste()	Вставляет содержимое системного буфера в элемент управления
PasteSpecial()	Вставляет содержимое системного буфера в элемент управления, используя заданный формат
ReplaceSel()	Заменяет выделенный текст данным текстом
RequestResize()	Принуждает элемент управления передать сообщение с уточнением EN_REQUESTRESIZE
SetBackgroundColor()	Устанавливает цвет фона для элемента управления
SetDefaultCharFormat()	Устанавливает формат символа, заданный по умолчанию
SetEventMask()	Устанавливает маску событий для элемента управления
SetModify()	Инвертирует флаг модификации текста
SetOLECallBack()	Устанавливает COM-объект IRichEditOleCallback для данного элемента
SetOptions()	Устанавливает опции для элемента управления
SetParaFormat()	Устанавливает формат выделенного абзаца
SetReadOnly()	Запрещает редактирование в элементе управления
SetRect()	Устанавливает прямоугольник форматирования для элемента
SetSel()	Устанавливает выделение текста
SetSelectionCharFormat()	Устанавливает формат символа, заданный для выделенного текста
SetTargetDevice()	Устанавливает выходное устройство-приемник для элемента управления
SetWordCharFormat()	Устанавливает формат для символов текущего слова
StreamIn()	Считывает текст из заданного потока

Функция	Назначение
StreamOut()	Считывает текст из заданного потока
Undo()	Выполняет откат на одну операцию редактирования текста

Работа с расширенным текстовым полем

Наше простое демонстрационное приложение может показать вам некоторые базовые приемы работы с элементом управления этого типа, а именно — как устанавливать атрибуты символов и формат абзаца. Включая расширенное текстовое поле в приложение, вы, естественно, захотите предоставить пользователю некоторые средства управления его работой. Поэтому, как правило, формируются меню команд и панель инструментов, с помощью которых можно будет выбрать те или иные опции, поддерживаемые в приложении. Но наше приложение Common не имеет ни меню, ни панели инструментов. Здесь для управления расширенным текстовым полем применяются четыре кнопки, на которых пользователь может щелкать и тем самым давать элементу управления некоторые команды.

Благодаря механизму карты сообщений, который поддерживается MFC, эти кнопки (точнее — события, с ними связанные) ассоциируются с функциями, которые отвечают за обработку событий. Добавьте в карту сообщений файла заголовка следующие операторы объявления обработчиков событий — щелчков на кнопках.

```
afx_msg void OnULine();
afx_msg void OnLeft();
afx_msg void OnCenter();
afx_msg void OnRight();
```

В карту сообщений файла реализации также добавьте операторы.

```
ON_COMMAND(IDC_RICEDIT_ULINE, OnULine)
ON_COMMAND(IDC_RICEDIT_LEFT, OnLeft)
ON_COMMAND(IDC_RICEDIT_CENTER, OnCenter)
ON_COMMAND(IDC_RICEDIT_RIGHT, OnRight)
```

Эти функции довольно просты. Добавьте их код в файл CommonView.cpp. Например, код функции OnULine() будет выглядеть следующим образом.

```
void CCommonView::OnULine()
{
  CHARFORMAT charFormat;
  charFormat.cbSize = sizeof(CHARFORMAT);
  charFormat.dwMask = CFM_UNDERLINE;
  m_richEdit.GetSelectionCharFormat(charFormat);

  if (charFormat.dwEffects & CFM_UNDERLINE)
    charFormat.dwEffects = 0;
  else
     charFormat.dwEffects = CFE_UNDERLINE;

  m_richEdit.SetSelectionCharFormat(charFormat);
  m_richEdit.SetFocus();
}
```

Функция OnULine() создает и инициализирует экземпляр структуры CHARFORMAT, который хранит информацию о форматировании символов. Текст объявления этой структуры представлен в листинге 10.13.

Листинг 10.13. Структура CHARFORMAT, объявленная в MFC

```
typedef struct _charformat
{
  UINT     cbSize;
  _WPAD     _wPad1;
  DWORD    dwMask;
  DWORD    dwEffects;
  LONG     yHeight;
  LONG     yOffset;
  COLORREF  crTextColor;
  BYTE     bCharSet;
  BYTE     bPitchAndFamily;
  char     szFaceName[LF_FACESIZE];
  _WPAD      _wPad2;
} CHARFORMAT;
```

В структуре CHARFORMAT член cbSize — это размер структуры, член dwMask указывает, какие члены данного экземпляра структуры содержат данные, а какие не принимаются во внимание. Значение dwMask может быть комбинацией символов констант CFM_BOLD, CFM_COLOR, CFM_FACE, CFM_ITALIC, CFM_OFFSET, CFM_PROTECTED, CFM_SIZE, CFM_STRIKEOUT и CFM_UNDERLINE. Член dwEffects определяет эффекты отображения символов. Значение dwEffects может быть комбинацией символов констант CFE_AUTOCOLOR, CFE_BOLD, CFE_ITALIC, CFE_STRIKEOUT, CFE_UNDERLINE и CFE_PROTECTED. Значение члена yHeight задает высоту символов, члена yOffset — смещение базового уровня для под- и надстрочных символов, члена crTextColor — цвет символов, члена bCharSet — набор символов (см. описание члена ifCharSet структуры LOGFONT), члена bPitchAndFamily — шаг набора символов (pitch) и его семейство, члена szFaceName — имя шрифта.

После инициализации экземпляра структуры CHARFORMAT программа вызывает метод GetSelectionCharFormat класса CRichEditCtrl. Эта функция имеет единственный аргумент — ссылку на экземпляр структуры CHARFORMAT, которая заполняется данными о формате символа. OnULine() анализирует значение члена dwEffects для того, чтобы определить, что нужно делать с подчеркиванием — включать или выключать. Терминальный оператор побитового И — & — используется для того, чтобы выделить одиночный бит переменной.

И наконец, после установки формата символа функция OnULine() возвращает фокус ввода в расширенное текстовое поле. Пользователь в момент щелчка на кнопке перевел на нее фокус ввода, а теперь программа должна вернуть его в текстовое поле, вызвав метод SetFocus() класса CRichEditCtrl. Иначе пришлось бы это делать самому пользователю — щелкнуть на текстовом поле.

Приложение Common также позволяет пользователю выбирать между тремя способами выравнивания абзаца. Эта операция программируется по той же методике, что и управление форматом символа. Тексты функций OnLeft(), OnRight() и OnCenter() представлены в листинге 10.14. Его нужно добавить в файл CommonView.cpp. Эти функции организуют соответствующее выравнивание абзаца. Как видно из листинга, наиболее существенная разница между этими функциями и OnULine()состоит в том, что используется структура PARAFORMAT вместо CHARFORMAT и вызов функции SetParaFormat() вместо SetSelectionCharFormat().

Листинг 10.14. Файл CommonView.cpp — функция изменения формата абзаца

```
void CCommonView::OnLeft()
{
   PARAFORMAT paraFormat;
   paraFormat.cbSize = sizeof(PARAFORMAT);
   paraFormat.dwMask = PFM_ALIGNMENT;
```

```
  paraFormat.wAlignment = PFA_LEFT;
  m_richEdit.SetParaFormat(paraFormat);
  m_richEdit.SetFocus();
}

void CCommonView::OnCenter()
{
  PARAFORMAT paraFormat;
  paraFormat.cbSize = sizeof(PARAFORMAT);
  paraFormat.dwMask = PFM_ALIGNMENT;
  paraFormat.wAlignment = PFA_CENTER;
  m_richEdit.SetParaFormat(paraFormat);
  m_richEdit.SetFocus();
}

void CCommonView::OnRight()
{
  PARAFORMAT paraFormat;
  paraFormat.cbSize = sizeof(PARAFORMAT);
  paraFormat.dwMask = PFM_ALIGNMENT;
  paraFormat.wAlignment = PFA_RIGHT;
  m_richEdit.SetParaFormat(paraFormat);
  m_richEdit.SetFocus();
}
```

Теперь можно вновь оттранслировать приложение и запустить его на выполнение. Попробуйте поработать с расширенным текстовым полем. Сначала щелкните мышью на этом элементе, чтобы переместить на него фокус ввода. Затем просто начните набирать какой-либо текст на клавиатуре. Можно также "пощупать" атрибуты текста. Для этого щелкните на кнопке ULine. Тем самым вы либо установите режим подчеркивания для далее вводимого текста, либо подчеркнете выделенный в поле текст. Для форматирования абзаца щелкните на кнопке Left, Center или Right — и абзац будет выровнен соответственно. На рис. 10.9 показано расширенное текстовое поле , в котором использованы различные стили оформления символов и абзацев.

Элемент формирования IP-адреса

Если вам приходилось заниматься разработкой приложений, работающих с сетью Internet, вы, вероятно, сталкивались с задачей проверки достоверности ввода соответствующей информации пользователем. В частности, речь могла идти о проверке правильности ввода IP-адреса (адреса Internet-протокола), например такого:

```
205.210.40.1
```

IP-адреса всегда должны состоять из четырех чисел, разделенных символом *точки*, причем каждое из них должно находиться в диапазоне от 1 до 255. Элемент управления ввода IP-адреса гарантирует, что введенная пользователем информация будет полностью соответствовать этому формату.

Чтобы опробовать на практике такой элемент, включите в текст метода OnCreate() оператор вызова функции CreateIPAddress(). Добавьте код этого метода в класс представления нашего приложения Common. Данный код на удивление прост — это вызов метода Create() класса элемента управления CIPAddressCtrl:

```
void CCommonView::CreateIPAddress()
{
  m_ipaddress.Create(WS_CHILD | WS_VISIBLE | WS_BORDER,
     CRect(520,40,700,65), this, IDC_IPADDRESS);
}
```

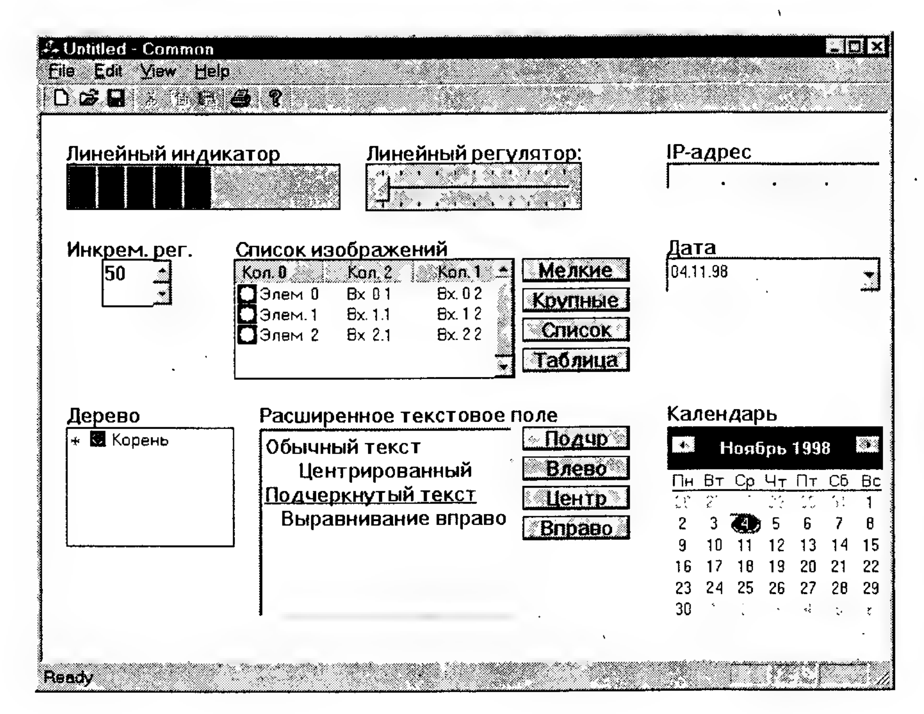


Рис. 10.9. Расширенное текстовое поле является, по сути, полнофункциональным текстовым процессором

Не забудьте определить идентификатор ресурса `IDC_IPADDRESS`. Для элемента управления этого типа специальные стили не определены. В состав класса входит несколько методов, которые позволяют установить значение параметра, очистить установленное ранее значение или каким-либо образом его отредактировать. Подробно с ними можно ознакомиться в электронной справке Visual C++.

Оттранслируйте новый вариант приложения и поработайте с элементом формирования IP-адреса. Попробуйте ввести в поля адреса цифры и буквы. Обратите внимание, что элемент сразу реагирует на ошибочный ввод (попробуйте, например, ввести в одно из полей число 999). Конечно, этот элемент достаточно прост, но если в приложении необходимо запрашивать у пользователя подобную информацию, включение его в программу окажется весьма кстати.

Элемент управления для работы с датами

Как часто встречаются приложения, в которых пользователю предлагается ввести дату? И какое раздражение всегда вызывает у пользователей необходимость вручную вводить дату в некотором предопределенном формате. Одни в таком случае предпочитают вызвать на экран календарь и выбрать необходимую дату из него. Другие полагают такой метод слишком медленным и скорее согласятся ввести значение, особенно если это можно сделать, откорректировав имеющуюся заготовку. Элемент для работы с датами, который поддерживает класс `CDateTimeCtrl`, удовлетворит и тех, и других.

Начнем, как обычно, с того, что добавим вызов `CreateDatePicker()` в метод `CCommonView::OnCreate()`, а затем добавим эту функцию в класс `CCommonView` в качестве локального метода. Также нужно добавить определение идентификатора ресурса `IDC_DATE`. Как

и при включении в приложение элемента формирования IP-адреса, единственное, что требуется выполнить в CreateDatePicker(), — это вызвать метод Create() класса элемента управления.

```
void CCommonView::CreateDatePicker()
{
  m_date.Create(WS_CHILD | WS_VISIBLE | DTS_SHORTDATEFORMAT,
    CRect(520,120,700,150), this, IDC_DATE);
}
```

В классе CDateTimeCtrl, объектом которого является m_date, определены специальные константы стилей, которые можно использовать при создании экземпляра класса. В табл. 10.11 перечислены эти константы и даны пояснения относительно их назначения.

Таблица 10.11. Константы стиля для элемента ввода даты

Стиль	Назначение
DTS_APPCANPARSE	Настраивает элемент ввода даты таким образом, что он передает часть функций управления приложению в то время, когда пользователь вводит дату
DTS_LONGDATEFORMAT	Выводит дату в расширенном формате соответственно настройке локализованной версии операционной системы, например Monday, May 18, 1998 в версии для США
DTS_RIGHTALIGN	Выравнивает выведенную информацию по правой границе поля (если этот стиль не будет задан, выполняется выравнивание по левой границе)
DTS_SHOWNONE	Вывод даты является необязательной функцией; настройка вывода даты выполняется специальным флажком
DTS_SHORTDATEFORMAT	Выводит дату в сокращенном формате соответственно настройке локализованной версии операционной системы, например 5/18/98 в версии для США
DTS_TIMEFORMAT	Помимо даты, выводит время
DTS_UPDOWN	Для ввода даты вместо календаря используется инкрементный регулятор

Среди многочисленных методов класса CDateTimeCtrl, которые позволяют настраивать цвет и шрифт выведенных данных, важнейшее место принадлежит функции GetTime(). Этот метод позволяет считать установленные пользователем время и дату. Метод формирует объект класса COleDateTime или CTime либо структуру SYSTEMTIME, члены которой доступны всем функциям приложения. Ниже представлено объявление структуры SYSTEMTIME.

```
typedef struct _SYSTEMTIME
{
   WORD  wYear
   WORD  wMonth
   WORD  wDayOfWeek
   WORD  wDay
   WORD  wHour
   WORD  wMinute
   WORD  wSecond
   WORD  wMilliseconds
} SYSTEMTIME;
```

Работать с данными, представленными в объекте такой структуры, легче, чем с объектом класса CTime. Этот класс описан в приложении E.

Теперь самое время на практике проверить, насколько удобно работать с элементом ввода даты. Оттранслируйте приложение и запустите его на выполнение. Щелкните на кнопке раскрытия рядом с полем представления даты в сокращенном формате — и откроется календарь

месяца. Выберите в нем нужную дату (при этом в основном поле данные изменятся соответственно вашему выбору). Теперь измените значение месяца в основном поле. Снова раскройте календарь, и вы увидите, что он представляет дни нового месяца, а подсветка сдвинулась именно на ту дату, которая установлена в основном поле. Обратите внимание, что в календаре число текущей даты обведено контуром.

Сам по себе календарь представляет собой отдельный элемент управления, который можно включить в приложения независимо от элемента ввода даты. Технология программирования календаря будет описана в следующем разделе.

Календарь

Добавим вызов функции CreateMonth() в метод CCommonView::OnCreate(), а затем добавим эту функцию в класс CCommonView в качестве локального метода. Также нужно добавить определение идентификатора ресурса IDC_MONTH. Текст метода CreateMonth() приведен ниже.

```
void CCommonView::CreateMonth()
{
  m_month.Create(WS_CHILD | WS_VISIBLE | DTS_SHORTDATEFORMAT,
    CRect(520,260,700,420), this, IDC_MONTH);
}
```

При создании этого элемента управления можно использовать большинство описанных выше стилей DTS_, но в классе CMonthCalCtrl, объектом которого является m_month, определены специальные константы стилей, которые можно использовать при создании экземпляра класса. В табл. 10.12 перечислены эти константы и даны пояснения относительно их назначения.

Таблица 10.12. Константы стиля для календаря

Стиль	Назначение
MCS_DAYSTATE	Настраивает элемент управления таким образом, что он посылает приложению сообщение MCN_GETDAYSTATE с тем, чтобы отдельные даты (например, праздники) отображались полужирным шрифтом
MCS_MULTISELECT	Позволяет пользователю выбирать некоторый интервал дат
MCS_NOTODAY	Подавляет вывод текущей даты в нижней части окна элемента. Пользователь может запросить вывод текущей даты, щелкнув на Today
MCS_NOTODAY_CIRCLE	Подавляет вывод контура вокруг текущей даты
MCS_WEEKNUMBERS	Выводит номер недели в пределах года в диапазоне от 1 до 52 в левом столбце календаря

Методы класса CMonthCalCtrl позволяют настраивать цвет и шрифт вывода, день (воскресенье или понедельник), с которого начинается отсчет очередной недели, и т.д. Наибольший интерес представляет функция GetCurSel(), которая формирует объект класса COleDateTime или CTime либо структуру LPSYSTEMTIME соответственно выбранной пользователем дате.

Оттранслируйте приложение и запустите его на выполнение. Поэкспериментируйте с календарем. Перейдите от одного месяца к другому. Если вы зайдете довольно далеко от текущей даты, вернуться к текущему месяцу можно будет, щелкнув на Today в нижней строке. Этот элемент управления составит достойную конкуренцию многочисленным аналогам от сторонних разработчиков, которые приходилось ранее включать в приложения.

Прокрутка изображения

После того как все перечисленные в этой главе элементы управления будут включены в приложение Common, их изображения явно выйдут за границы видимой части окна. Как видно на рис. 10.9 в окне приложения отсутствуют полосы прокрутки, несмотря на то что класс CCommonView является наследником CScrollView. Для того чтобы механизм прокрутки заработал, необходимо соответствующим образом его настроить.

Разверните класс CCommonView в окне ClassView и дважды щелкните на методе OnInitialUpdate(). Отредактируйте текст приведенной ниже функции.

```
void CCommonView::OnInitialUpdate()
{
   CSrollView::OnInitialUpdate();

   CSize sizeTotal;
   sizeTotal.cx = 720;
   sizeTotal.cy = 500;

   SetScrollSizes(MM_TEXT, sizeTotal);
}
```

Поле последнего элемента управления, который мы добавили в предыдущем разделе (календаря), простирается от точки (520, 260) до (700, 420). В функции мы определили, что общий размер окна — 720×500 пикселей. Таким образом, между полем элемента и границей окна имеется достаточный зазор. Если видимый размер окна менее 720×500, появятся полосы прокрутки. Если видимый размер больше заданного, полосы прокрутки автоматически исчезнут. Вызов SetScrollSizes() передает этой функции все заботы, связанные с созданием полос прокрутки, установкой их размеров соответственно соотношению видимой части и всего представления, а также реакцией на манипуляции пользователя полосами прокрутки. После повторной трансляции приложения и его запуска на выполнение опробуйте в работе созданные и настроенные средства прокрутки (рис. 10.10). А теперь вас ждет ответ на вопрос "Почему же ранее, до редактирования функции OnInitialUpdate(), полосы прокрутки не появлялись в окне приложения?". Дело в том, что в заготовке этой функции мастер AppWizard устанавливает размер поля представления приложения равным 100×100 пикселей, а при такой исходной настройке полосы прокрутки появятся только в случае, если размер клиентской области окна будет меньше 100×100.

Опробовав полосы прокрутки в действии, вы можете обнаружить, что приложение неверно реагирует на попытки манипулировать горизонтальной полосой. Причину можно найти, если воспользоваться методикой отладки программ, описанной в приложении Г. Все дело в функции OnHScroll(), которая полагает, что любая манипуляция горизонтальной полосой прокрутки, вызывающая сообщение WM_HSCROLL, относится к элементу управления типа линейный регулятор — объекту m_trackbar (см. листинг 10.4). Текст этой функции нужно отредактировать следующим образом.

```
void CCommonView::OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
  CSliderCtrl* slider = (CSliderCtrl*)pScrollBar;
  if (slider == &m_trackbar)
   {
     int position = slider->GetPos();
     char s[10];
     wsprintf(s, "%d  ", position);
     CClientDC clientDC(this);
     clientDC.TextOut(390, 22, s);
   }
  CScrollView::OnHScroll(nSBCode, nPos, pScrollBar);
}
```

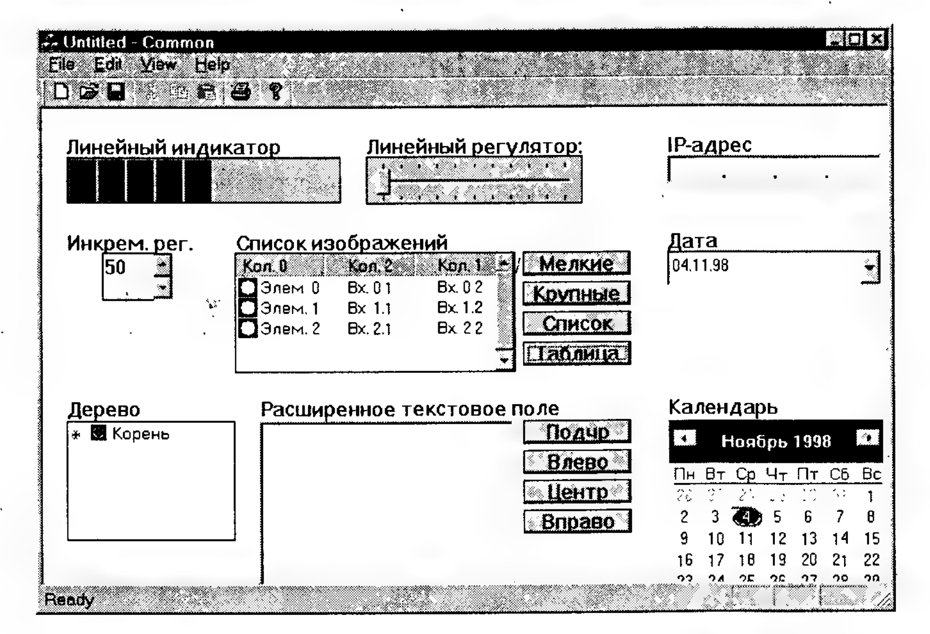


Рис. 10.10. Окно приложения Common после настройки полос прокрутки

Теперь та часть программы, которая относится к обслуживанию линейного регулятора, будет выполняться только в случае, если сообщение послано объектом `m_trackbar`. В остальных случаях управление сразу передается методу `OnHScroll()` базового класса.

После этой коррекции снова оттранслируйте и запустите приложение. На сей раз оно должно работать, как часы!

ГЛАВА 11

Справка в приложении

В этой главе...

Типы справочных систем

Компоненты справочной системы

Поддержка справочной системы, предоставляемая мастером AppWizard

Планирование структуры справочной системы

Создание системы командной справки

Создание системы контекстной справки

Подготовка справочных текстов

Реорганизация оглавления справочной системы

Многие программисты отказываются от создания в приложениях какой бы то ни было справочной системы. Но даже те, кто добавляют справочную систему в свои приложения, откладывают эту процедуру на самый конец разработки. Но что можно создать, когда отпущенное на проект время почти исчерпано? Уже не остается времени на обдумывание справочных текстов или разработку подпрограмм, организующих выдачу этого текста по запросам пользователя. Одной из причин такого положения дел является уверенность большинства разработчиков в том, что создание справочной системы — занятие очень сложное и трудоемкое. Но при работе с Visual C++ все обстоит гораздо проще, чем вы думаете. Visual C++ даже способен написать вместо вас некоторую часть справочных текстов! В этой главе описывается включение справочной системы в приложение ShowString, которое мы создали в главе 8.

Типы справочных систем

Справочные системы можно классифицировать по-разному. В данном разделе системы справки анализируются по четырем критериям.

- Каким образом пользователь ее вызывает
- Как выглядит окно справки на экране
- Ответ какого типа хочет получить пользователь
- Каким образом разработчик включает справочную систему в текст программы

Ни на один из этих вопросов нельзя дать однозначного ответа. Существует по крайней мере девять способов обращения к системе справки, три стандартных варианта отображения окна справки и три различных задачи, которые нужно решить, включая справочную систему в приложение. Все эти подходы к анализу справочных систем должны помочь вам понять, почему существует несколько технологий включения справочных систем в приложение, а также рассеять недоумение по этому поводу, которое может появиться при первом знакомстве с обсуждаемым предметом.

Доступ к справочной системе

Первое, что характеризует справочную систему, — это способ получения к ней доступа. Существует несколько способов обращения к системе справки.

- Выбор команды меню Help (Справка), например Help⇨Topics (Тема справки). (Выбор команды What's This (Что это?) или About (О программе) не вызовет немедленного обращения к справочной системе.)
- Нажатие клавиши <F1>.
- Щелчок на кнопке Help в диалоговом окне.
- Щелчок на пиктограмме What's This (Что это?) панели инструментов с последующим щелчком на каком-либо ином элементе.
- Выбор команды What's This меню Help (или меню System для приложений, использующих диалоговые окна) и последующий щелчок на интересующем вас объекте.
- Щелчок в диалоговом окне на пиктограмме Question (Вопрос) с последующим щелчком на одном из элементов окна.
- Щелчок где-либо правой кнопкой мыши с последующим выбором из раскрывшегося контекстного меню команды What's This.
- Нажатие в некоторых старых приложениях клавиш <Shift+F1> и последующий щелчок на интересующем вас элементе.

■ Двойной щелчок на файле с расширением HLP без запуска приложения.

В первых трех способах пользователь для обращения к системе справки выполняет всего лишь одно действие (выбирает команду меню, нажимает <F1> или щелкает на кнопке). Последующие пять способов предусматривают выполнение двух шагов: первый — чаще всего щелчок для перехода в состояние *вопроса* (формально он называется режимом Whats This), второй — указание на то, о чем требуется получить справку. Пользователи соответственно классифицируют справку как выполняемую за один шаг или за два шага.

На заметку

В среде Visual Studio организация системы справки отличается от описанной в этой глвве. Большая чвсть информвции в этой системе представляется в составе отдельного программного продуктв MSDN в формвте HTML, хотя в некоторых случвях будет выполнено обращение и к более трвдиционным системам спрввки. Для экспериментов с системой спрввки, описанных в этой глвве, воспользуйтесь простыми утилитвми и приложениями, поставляемыми в составе вашей оперционной системы, либо используйте возможности самой операционной системы. Стврые версии приложений, твкие квк Word и Excel, не смогут поддерживвть основные концвпции систем спрввки, принятые в Windows 95, поскольку последние значительно отличвются от концепций, которые были приняты рвнее.

Справка в формате HTML

До последнего времени все фвйлы справочных систем в приложениях строились нв бвзе расширенного текстового формата (RTF). В послвдних же приложениях Microsoft с этой целью нвчала использовать формат фвйлов HTML и выпустилв нв рынок множество программных продуктов, упрощающих процесс подготовки подобных спрввочных систем.

Применение нового форматв имеет ряд серьезных преимущвств. В чвстности, файлы спрввки могут включвть связи с ресурсами, доступными по сети Internet. В соствв спрввки можно включать любые вктивные компоненты, которые воспринимвются броузером, в том числе и элементы упрввления ActiveX, вплеты Javв и рвзличного родв сценврии. Среди рвзрвботчиков все более ширится мнение, что использовввние формвтв HTML для спрввочной системы знвчительно сокращвет время рвзрвботки.

Но, квк часто бывввет, применение HTML имеет и обрвтную сторону. Нвпример, интерфейс с системой справки не так богвт, как при использоввнии традиционных формвтов. Поэтому многие рвзрвботчики, вдоволь нвигрввшись с HTML-документвми в составе спрввочной системы Visubl Studio, не очень охотно используют подобную методику в собственных приложениях.

Если вместо трвдиционной системы спрввки, о которой пойдет речь в этой глвве, вы захотите использоввть HTML-документы, нвчните с посещения Web-стрвницы **http://www.microboft.com/workshop/buthor/htmlhelp** и скопируйте Help Workshop, который включвет множество документов и примеров.

В общих чертвх процесс создвния HTML-спрввки во многом вналогичен тому, который будет описвн ниже. Отличия включвют, в частности, использоввние функции HTMLHelp() вместо ::WinHelp(), в твкже редвктироввние HTML-документов при помощи HTML Hвlp Workshop вместо рвботы с RTF-фвйлами в среде MS Word.

Представление справочной информации на экране

Второй метод классификации систем справки основан на том, как окно справки представлено на экране. Существует несколько способов отображения справочной информации.

■ **Диалоговое окно** Help Topics. Это диалоговое окно, показанное на рис. 11.1, дает возможность пользователю провести поиск в индексе, просмотреть оглавление или найти в справочных текстах требуемое слово.

■ **Обычное окно** Help. В этом окне, показанном на рис. 11.2, содержатся кнопки Help Topics, Back и Options. Размер этого окна можно изменить: минимизировать, максимизироввть или закрыть. Чаще всего оно находится поверх других окон, так же, как системные часы или окна других популярных утилит.

- **Всплывающее окно.** Пример окна этого типа приведен на рис. 11.3. Всплывающее окно имеет относительно небольшие размеры и не содержит никаких кнопок или меню. Такие окна закрываются, когда вы щелкаете где-либо вне этого окна, их нельзя минимизировать или перемещать. В подобных окнах выводится справочная информация небольшого объема или краткие пояснения.

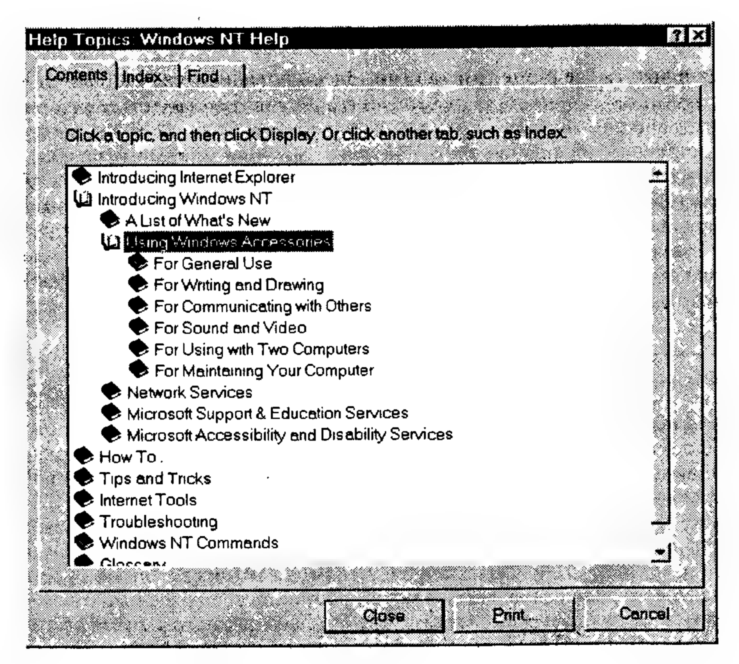


Рис. 11.1. В диалоговом окне Help Topics *пользователю предоставляется возможность просмотреть оглавление, выполнить поиск по индексу или найти тексты справки, содержащие заданное слово*

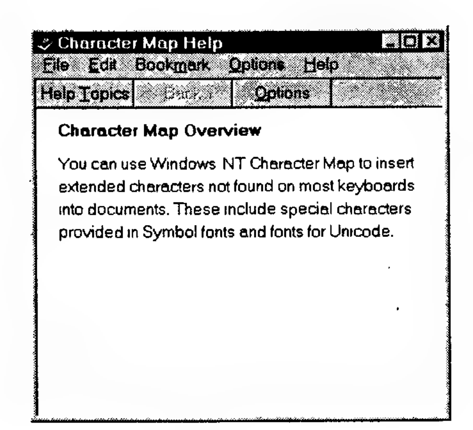


Рис. 11.2. Обычное окно Help *содержит кнопки и может иметь меню. Оно допускает выполнение любых действий, которые можно применять к окнам других типов*

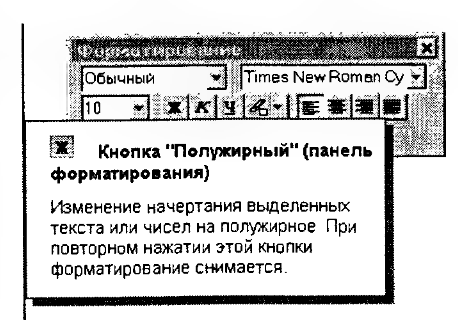


Рис. 11.3. Всплывающие окна Help *предоставляют пользователю гораздо меньше возможностей управления и применяются только для предоставления кратких пояснений*

Назначение справочной информации

Третий метод классификации справочных систем основывается иа анализе причин, по которым пользователь обращается к данной справочной системе. В книге *The Windows Interface Guidelines for Software Design* Microsoft выделяет справочные системы следующих типов.

- **Контекстная справка**. В подобных системах предусматриваются ответы на вопросы типа "Для чего предиазначена данная кнопка?" и "Что означает данная опция?".
- **Справка о методике решения задачи**. Здесь объясняется, как выполнить то или иное задание, например распечатать документ (часто процедура может состоять из нескольких этапов).
- **Справка по ссылке**. Здесь выполняется поиск описания параметра функции или имени шрифта, или любых других данных, которые могут время от времени потребоваться опытному пользователю.
- **Мастера** последовательно проводят пользователя через все стадии решения сложной задачи подобно тому, как AppWizard проводит пользователя Visual Studio через стадии настройки проекта.

Все вышеизложенное описывает содержание данных, предоставляемых пользователю. Хотя подобные описания важны для проектировщика систем справки и авторов ее текстов, с точки зрения программиста, они не имеют большого значения.



Книга, о которой шла речь вышв, входит в состав продукта MSDN и распространяется в электронном формате вместе с Visual Studio.

Программирование справочных систем

Последний и, пожалуй, самый важный, с точки зрения разработчика, метод классификации справочных систем основан на анализе вариантов построения текста программы справочной системы. В Windows существует три типа сообщений, посылаемых в тот момент, когда пользователь обращается к справочной системе одним из возможных способов.

- WM_COMMAND
- WM_HELP
- WM_CONTEXTMENU

На заметку

Сообщения Windows обсуждались в главе 3.

Когда пользователь выбирает в меню команду Help или щелкает на кнопке Help в диалоговом окне, система, как и всегда, посылает сообщение WM_COMMAND. Для отображения соответствующей справки нужно перехватить это сообщение и вызвать систему WinHelp.

Когда пользователь выполняет щелчок правой кнопкой мыши на некотором элементе вашего приложения, посылается сообщение WM_CONTEXTMENU. Нужно перехватить это сообщение и построить в данном месте контекстное меню. Поскольку в большинстве случаев ваше контекстное меню будет содержать только комаиду What's This, можно воспользоваться заранее подготовленным меню, содержащим только этот элемент, и делегировать отображение меню

системе справки. Подробнее эта процедура обсуждается ниже, в разделе *Создание системы контекстной справки.*

Когда пользователь обращается к справочной системе любым другим способом, большая часть работы выполняется встроенными обработчиками. Вам не потребуется перехватывать сообщения, переводящие приложение в режим Whats This, изменять вид курсора и обрабатывать щелчки, выполняемые пользователем в этом режиме. Нужно лишь перехватить сообщение WM_HELP, идентифицирующее элемент управления, диалоговое окно или меню, о котором затребована справка, а затем предоставить эту справку. Нажал ли пользователь клавишу <F1> или перевел приложение в режим Whats This с последующим выбором элемента, значения не имеет. Фактически в приложении вы даже не сможете обнаружить различий в этих действиях.

Сообщения WM_HELP и WM_CONTEXTMENU обрабатываются почти идентично, поэтому, с точки зрения разработчика, существует только два типа справочных систем. Мы будем называть их *командной справкой* и *контекстной справкой*. Каждый из этих типов справки будет обсуждаться ниже в данной главе, в разделах *Создание системы командной справки* и *Создание системы контекстной справки* соответственно. Однако не забывайте, что нет никакой связи между разделением справочных систем на эти два типа и придуманного пользователями разделения справочных систем на справки, получаемые за один шаг и за два шага.

Компоненты справочной системы

Как вы могли предполагать, при создании интерактивной справочной системы используется большое количество различных файлов. Конечным продуктом, который будет поставляться заказчикам, является файл справки, имеющий расширение .hlp. Он создается на основе нескольких файлов. В приведенном ниже списке значение appname следует замеинить именем EXE-файла вашего приложения. Если имя не будет указано, может быть создано несколько файлов с различными именами. Мастер AppWizard создает следующие файлы — компоненты справочной системы.

.h	Эти файлы заголовков (Header files) содержат определения идентификаторов ресурсов и идентификаторы тем справки, которые будут использоваться в программах на C++
.hm	Эти файлы адресации справок (Help Mapping files) содержат идентификаторы тем справки. Файл appname.hm генерируется всякий раз, когда вы компилируете приложение (не следует вручную вносить в него изменения)
.rtf	Эти файлы расширенного текстового формата (Rich Text Format files) содержат тексты справок по каждой теме
appname.cnt	Этот файл таблицы содержания используется для подготовки вкладки Contents (Оглавление) в диалоговом окне Help Topics. (Нужно будет поставлять этот файл оглавления вместе с приложением в дополнение к файлу .hlp.)
appname.hpj	Этот файл проекта справочной системы (Help Project File) объединяет файлы .hm и .rtf, совместно используемые при компиляции файла .hlp

В процессе использования справочная система создает файлы и других типов. При выполнении деинсталляции приложения обязательно проведите поиск и удаление всех файлов указанных ниже типов, которые могли быть созданы в дополнение к файлу .hlp.

- appname.gid (файл конфигурации, как правило, скрытый)
- appname.fts (файл поиска по всему тексту, который генерируется, когда пользователь выполняет команду Find в текстах справки)
- appname.ftg (групповой список поиска по всему тексту, который также генерируется, когда пользователь выполняет команду Find)

Идентификаторы справочных тем связывают справочные тексты со справочной системой. Ваша программа управляет справочной системой, передавая ей идентификатор нужной темы для ее отображения на экране, например HID_FILE_OPEN. Справочная система выполняет поиск этого идентификатора темы в файле .hlp, полученном при компиляции файлов .rtf, включая и тот rtf-файл, в котором содержался текст справки с требуемым идентификатором. (Этот процесс проиллюстрирован на рис. 11.4.) Каждый идентификатор темы справки должен быть определен дважды: один раз для использования системой справки и второй раз — для использования в вашей программе. Когда система справки отображает на экране справочную тему или диалоговое окно Help Topics, она самостоятельно выполняет поиск и вывод на экран других справочных тем, если пользователь этого затребует.

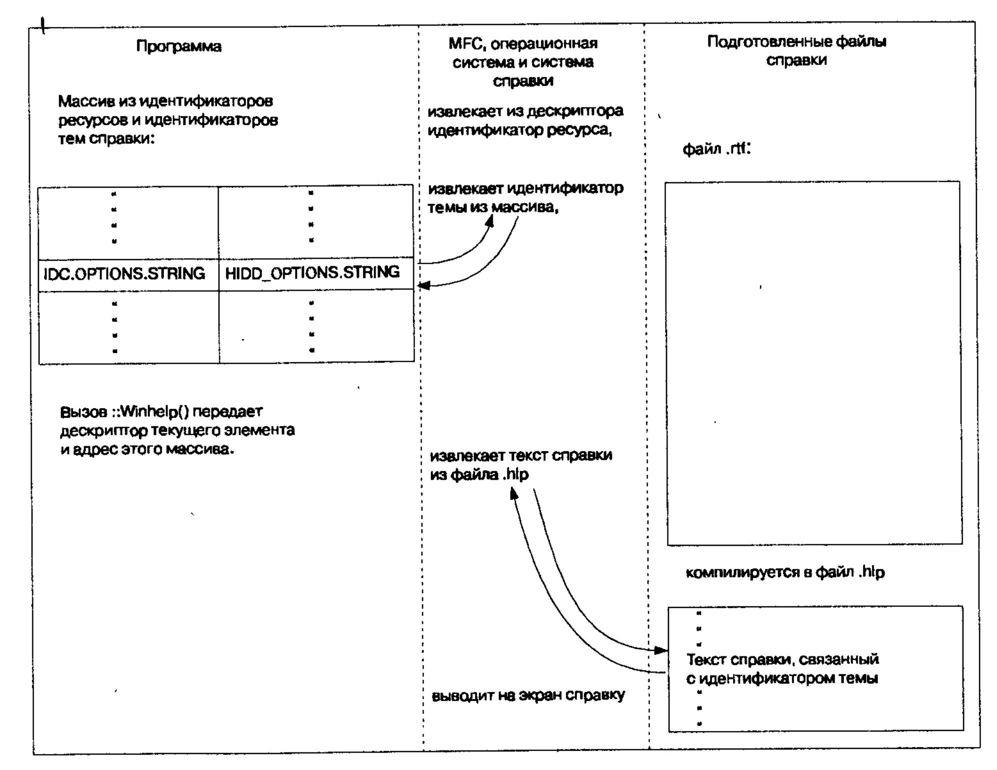


Рис. 11.4. Для отображения справочной темы необходима совместная работа программы, справочной системы и подготовленных вами справочных файлов

Поддержка справочной системы, предоставляемая мастером AppWizard

При создании с помощью мастера AppWizard приложения с многооконным интерфейсом (но без поддержки баз данных и OLE) и выборе опции Context Sensitive Help (Контекстно-зависимая справка) будет выполнено следующее.

- В карту сообщений будут добавлены элементы для перехвата команд ID_HELP_FINDER, ID_HELP, ID_CONTEXT_HELP и ID_DEFAULT_HELP. Никакие подпрограммы обработки этих сообщений добавлены не будут, они будут передаваться функциям-членам класса CMDIFrameWnd.
- На панель инструментов будет помещена пиктограмма What's This.
- Команда Help Topics будет добавлена в оба типа меню Help, генерируемых мастером AppWizard (меню, используемом, когда файл открыт, и меню меньшего размера, используемом, когда не открыт ни один файл).
- Будут добавлены акселераторы для клавиш <F1> (ID_HELP) и <Shift+F1> (ID_CONTEXT_HELP).
- Сообщение Ready (Готово), помещаемое в строку состояния по умолчанию, будет заменено сообщением For Help, press F1 (Для получения справки нажмите F1).
- Будет добавлено новое приглашение Select an object on which to get Help (Укажите объект, по которому хотите получить справку), выводимое в строку состояния в режиме Whats This.
- К сообщениям, выводимым в строку состояния, будут добавлены сообщения меню Help и его команд.
- В проект включается копия файла afxcore.rtf, текстового файла справки для стандартных команд меню, таких как File⇨Open и т.д.
- В проект включается копия файла afxprint.rtf, текстового файла справки по выполнению печати (команда File⇨Print) и предварительному просмотру печатного документа (команда File⇨Print Preview). (Эти файлы добавляются к проекту отдельно, поскольку не во всех проектах могут использоваться вывод на печать и предварительный просмотр печатных документов. Если проект предусматривает работу с базами данных и использование средств OLE, в него будут включены дополнительные справочные файлы.)
- В проект будут включены копии двадцати двух файлов графических изображений в формате .bmp, используемые в качестве иллюстраций в справочных темах, посвященных командам File⇨Open и т.д.

После создания столь прочного фундамента задача построения в приложении справочной системы выполняется в три этапа.

1. Необходимо подготовить проект системы справки. Вы намерены предоставить материал только для ссылочной справки, только инструкции по решению конкретных задач либо и то, и другое одновременно? Каков будет размер текстов, подготовленных для вывода в контекстные всплывающие меню?
2. Необходимо создать программные "крючки", которые будут вызывать отображение подготовленных справочных тем. Эта процедура выполняется отдельно для командной и контекстной справок, как будет показано в последующих разделах.

3. Необходимо создать файлы .rtf, которые будут содержать темы справки по данному приложению и соответствующие идентификаторы тем справки. Если проект справочной системы был подготовлен тщательно и с учетом всех особенностей приложения, это не вызовет больших затруднений, хотя и потребует определенных затрат времени.

На заметку В больших проектах тексты справочной системы часто готовятся не программистами, а специалистами по подготовке технической документации. В этой ситуации необходима тщательная координация действий. Вам потребуется предоставить авторам справочных текстов идентификаторы справочных тем и, возможно, объяснить некоторые функции так, чтобы они могли быть описаны в справочной системе. Работу эту необходимо проводить в тесном контакте друг с другом, с уважением относясь к знаниям и опыту коллег.

Планирование структуры справочной системы

Процесс разработки справочной системы напоминает процесс разработки программного обеспечения. Его не следует выполнять, не имея четкого плана действий. А кроме того, не следует оставлять эту работу на последний момент. Знаменитый опыт, проведенный многие десятилетия назад, предусматривал разделение программистов на две группы. От первой потребовали подготовить полное руководство по использованию программы еще до начала создания ее текста, а от второй группы программистов требовалось завершить разработку программы, прежде чем приступить к подготовке документации. Программы, разработанные первой группой, оказались лучше. Этим программистам удалось обнаружить ошибки проектирования на самой ранней стадии разработки проекта, еще до того, как они были зафиксированы в текстах программ. А кроме того, процесс программирования был проще.

Какой бы размер не имело ваше приложение, созданию для него справочной системы можно посвятить целую книгу. Если вам необходима дополнительная информация о том, как это сделать, обратитесь к книге *Designing Window 95 Help: A Guide to Creating Online Documents*, написанной Мэри Дитон (Mary Deaton) и Черил Локетт (Cheryl Lockett) и опубликованной издательством Que. В данной главе мы можем уделить внимаиие лишь нескольким основным рекомендациям.

В результате проектирования должен быть создан список справочных тем и обобщенная схема доступа к ним. Желательно, чтобы подготовленный вами перечень справочных тем включал следующее.

- Страницу или около того справочной информации по каждой команде каждого меню, доступ к которой можно будет получить в режиме Whats This после щелчка на соответствующей команде.
- Отдельную страницу, доступ к которой осуществляется через оглавление справки, содержащую перечень всех меню и всех входящих в них команд, с установленными связями со страницами, описывающими эти команды.
- Отдельную страницу, доступ к которой осуществляется через оглавление справки, для каждой из наиболее существенных задач, решаемых в данном приложении. Следует предусмотреть наличие примеров и пошаговых инструкций.
- Контекстную справку для элементов управления во всех диалоговых окнах.

Хотя все это может показаться вам работой непомерного объема, не забывайте, что справочный материал по всем основным элементам приложения автоматически предоставляется мастером AppWizard. Это относится к командам меню, стандартным диалоговым окнам и т.п.

После составления полного списка необходимых материалов и обобщенной схемы доступа к каждой странице справки следует подумать об установлении перекрестных связей между страницами. (Например, мастер AppWizard предоставляет вам справку по команде Open меню File, в которой имеются ссылки на описание команды New этого же меню, и наоборот.) Кроме того, полезно подготовить краткие определения для специфических терминов и ключевых слов.

В данном разделе мы подготовим схему справочной системы для приложения ShowString, созданного нами в главе 8. Это простое приложение предназначено для вывода на экран строки, определенной пользователем. Строка может быть центрирована по вертикали или по горизонтали, а также выведена черным, зеленым или красным цветом. В приложении имеется новое меню (Tools), содержащее единственную команду (Options). При ее выборе на экран выводится диалоговое окно, в котором пользователь может установить любой из указанных выше режимов. Проект справочной системы для этого приложения включает следующие элементы.

- Замена подготовленной мастером AppWizard строки-местодержателя строкой со значением ShowString или каким-либо иным, отражающим специфику приложения.
- Подготовка справочных тем, касающихся меню Tools и команды Options.
- Подготовка справочной темы по каждому из элементов управления, имеющихся в диалоговом окне Options.
- Добавление в диалоговое окно Options новой пиктограммы Question.
- Изменение предоставляемых мастером AppWizard справочных текстов, отображаемых в том случае, когда пользователь запросит контекстную справку по элементам окна.
- Подготовка текста и добавление в меню Help темы Understanding Centering (Понятие о центрировании).
- Реорганизация оглавления справочной системы для отражения нововведений.

Создание системы командной справки

С точки зрения разработчика, система командной справки относительно проста. (В любом случае вам потребуется написать объяснения, поэтому не расслабляйтесь.) Как мы уже знаем, мастер AppWizard добавляет в меню Help команду Help Topics и организует вход в карте сообщений для ее перехвата. Входящий в MFC класс CMDIChildFrame содержит метод, предназначенный для обработки этого сообщения. Так что в данном случае от вас не потребуется никаких действий. Однако, если вам понадобится добавить в меню Help какую-либо иную команду, нужно будет выполнить это точно так же, как и в случае любого другого меню: в окне ResourceView. Затем необходимо организовать в классе приложения (в данном случае — в CShowStringApp) перехват соответствующего сообщения.

В качестве примера поместим в приложение ShowString в меню Help новую команду с именем Understanding Centering. Вот как это делается.

1. Откройте в Visual Studio приложение ShowString — либо то, которое было создано в процессе работы над главой 8, либо его копию, позаимствованную с Web-страницы этой книги. Прежде чем продолжить, сделайте несколько копий этого проекта, поскольку в следующих главах мы будем использовать его в качестве отправной точки для дальнейшей работы.
2. Откройте меню IDR_MAINFRAME при помощи средств окна ResourceView — разверните пункт Menus и дважды щелкните на IDR_MAINFRAME. Добавьте пункт Understanding Centering в меню Help и оставьте то наименование идентификатора ресурса, которое предлагает Visual Studio (ID_HELP_UNDERSTANDINGCENTERING). Мы не будем сейчас отвлекаться на

мелочи, но это именно тот редкий случай, когда предлагаемое средой разработки наименование можно было бы и сократить.

3. Добавьте этот же пункт и в другое меню, IDR_SHORTTYPE, и оставьте тот же идентификатор ресурса.

4. С помощью мастера ClassWizard обеспечим в классе CShowStringApp перехват данного сообщения, как это обсуждалось ранее, в главе 8. Новая функция должна выглядеть следующим образом:

```
void CShowStringApp::OnHelpUnderstandingcentering()
{
  WinHelp(HID_CENTERING);
}
```

Этот единственный оператор запускает справочную систему и передает ей идентификатор темы справки HID_CENTERING. При выполнении компиляции этот идентификатор темы справки должен быть известен компилятору, поэтому поместим в файл ShowString.h следующую строку:

```
#define HID_CENTERING 0x01
```

Идентификаторы в диапазоне от 0x0000 до 0xFFFF зарезервированы для тем справки, определяемых пользователем. Поэтому значение 0x01 является вполне подходящим. Теперь все требования компилятора C++ удовлетворены, но при запуске приложения вызов функции WinHelp() не приводит к поиску темы, поясняющей, что такое центрирование. Вам необходимо добавить *вход адресации справки*. Это следует сделать в новом файле с именем ShowStringx.hm (x указывает на расширение, поскольку дополнительные входы адресации справок должны добавляться именно в дополнительный файл). Выберите команду File⇨New, в открывшемся диалоговом окне выберите вкладку Files, выделите значение Text file, введите в поле имени файла значение ShowStringx.hm и щелкните на кнопке OK. В новый файл поместите следующую строку:

```
HID_CENTERING 0x01
```

Сохраните новый файл. Далее вам следует отредактировать файл проекта справочной системы ShowString.hpj. Если вы сделаете двойной щелчок на имени этого файла в окне папки или в окне Windows 95 Explorer, файл будет открыт в окне Help Compiler. В нашем случае требуется отредактировать данный файл именно как текстовый, поэтому следует открыть его в приложении Visual Studio. В окне Project WorkSpace приложения Visual Studio выберите вкладку FileView, а затем откройте файл ShowString.hpj, дважды щелкнув на его имени в окне FileView (а вы удивлялись, для каких целей может служить эта вкладка), и добавьте в самый конец файла следующую строку:

```
#include <ShowStringX.hm>
```

После ввода этой директивы в текст файла ShowString.hpj обязательно нажмите <Enter>. В результате в файле образуется заключительная пустая строка, которая нужна для правильной работы компилятора справочной системы.

Теперь и справочная система, и компилятор имеют данные о новом идентификаторе темы справки. Ниже в данной главе, когда вы подготовите текст этой справки, мы добавим к справочной системе секцию, которая разъясняет суть процедуры центрирования, и свяжем ее с идентификатором темы справки.

Другим распространенным приемом использования командной справки является включение кнопки Help в диалоговые окна, что позволяет пользователю получить общее описание этого диалогового окна. Прежде данный подход считался стандартным приемом, но теперь

он рекомендуется только для сложных диалоговых окон, особенно для тех, в которых имеются сложные процедуры взаимодействия содержащихся в нем элементов управления. В подобном случае просто повторите все шаги, которые были выполнены во время добавления новой команды Understanding Centering в меню Help, но вместо команды добавьте кнопку. Не нужно создавать новый файл типа .hm. Поместите новую строку с идентификатором темы справки в уже созданный файл ShowStringX.hm, который будет продолжать расти по мере изучения следующих разделов этой главы.

Создание системы контекстной справки

Первым шагом в процессе создания контекстной справки для приложения ShowString будет размещение пиктограммы Question в диалоговом окне Options, поскольку AppWizard уже поместил соответствующую пиктограмму на панель инструментов. Откройте диалоговое окно Options, сделав двойной щелчок на его имени в окне ResourceView, и выберите команду View⇨Properties. В диалоговом окне Dialog Properties выберите вкладку Extended Styles и убедитесь, что флажок опции Context help установлен (рис. 11.5).

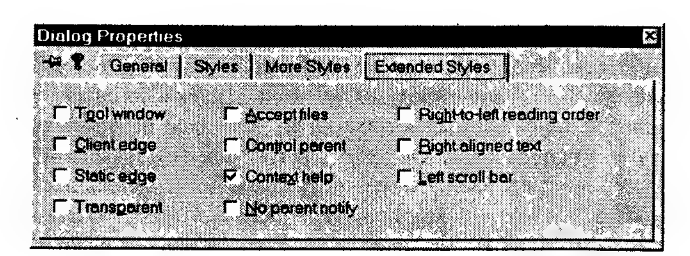


Рис. 11.5. Чтобы поместить кнопку Question в диалоговое окно Options приложения ShowString, необходимо установить флажок опции Context help на вкладке Extended Styles диалогового окна Dialog Properties

Как отмечалось выше, в контекстной справке используются два сообщения: WM_HELP, когда пользователь делает где-либо щелчок, причем ранее уже был установлен режим Whats This?, и сообщение WM_CONTEXTMENU, когда пользователь делает где-либо щелчок правой кнопкой мыши. В классе нашего диалогового окна COptionDialog необходимо организовать перехват этих сообщений. С этой целью поместим дополнительные входы в карту сообщений, добавив новые строки вне специального комментария, создаваемого ClassWizard. Теперь карта сообщений в файле OptionsDialog.h должна выглядеть так.

```
// Функции сгенерированной карты сообщений.
   //{{AFX_MSG(COptionsDialog)
      // Внимание: здесь будут размещены функции-члены,
      //  добавляемые ClassWizard.
   //}}AFX_MSG
   afx_msg BOOL OnHelpInfo(HELPINFO* lpHelpInfo);
   afx_msg void OnContextMenu(CWnd* pWnd, CPoint point);
    DECLARE_MESSAGE_MAP()
```

Карта сообщений в файле OptionsDialog.cpp будет выглядеть следующим образом.

```
BEGIN_MESSAGE_MAP(COptionsDialog, CDialog)
   //{{AFX_MSG_MAP(COptionsDialog)
      // ВНИМАНИЕ: здесь будут размещены макросы карты
      //  сообщений, добавляемые ClassWizard.
```

```
  //}}AFX_MSG_MAP
  ON_WM_HELPINFO()
  ON_WM_CONTEXTMENU()
END_MESSAGE_MAP()
```

Эти макросы организуют перехват сообщения WM_HELP функцией OnHelpInfo() и перехват сообщения WM_CONTEXTMENU функцией OnContextNenu(). Следующим шагом будет написание текста этих функций. Обе функции должны использовать таблицу для связывания идентификатора ресурса с идентификатором темы справки. Поместим соответствующие строки в начало файла OptionsDialog.cpp, сразу после комментария // COptionsDialog dialog.

```
static DWORD aHelpIDs[] =
{
  IDC_OPTIONS_STRING, HIDD_OPTIONS_STRING,
  IDC_OPTIONS_BLACK, HIDD_OPTIONS_BLACK,
  IDC_OPTIONS_RED, HIDD_OPTIONS_RED,
  IDC_OPTIONS_GREEN, HIDD_OPTIONS_GREEN,
  IDC_OPTIONS_HORIZCENTER, HIDD_OPTIONS_HORIZCENTER,
  IDC_OPTIONS_VERTCENTER, HIDD_OPTIONS_VERTCENTER,
  IDOK, HIDD_OPTIONS_OK,
  IDCANCEL, HIDD_OPTIONS_CANCEL,
  0, 0
};
```

Система справки будет использовать этот массив (его адрес вы передаете функции WinHelp()) для связывания идентификатора ресурса с идентификатором темы справки. Однако компилятор еще не имеет никаких сведений о HIDD_OPTIONS_STRING, поэтому добавьте соответствующие строки в файл OptionsDialog.h, поместив их до определения класса COptionsDialog.

```
#define HIDD_OPTIONS_STRING 2
#define HIDD_OPTIONS_BLACK 3
#define HIDD_OPTIONS_RED 4
#define HIDD_OPTIONS_GREEN 5
#define HIDD_OPTIONS_HORIZCENTER 6
#define HIDD_OPTIONS_VERTCENTER 7
#define HIDD_OPTIONS_OK 8
#define HIDD_OPTIONS_CANCEL 9
```

Числа задаются произвольно. Теперь требования компилятора выполнены, так как мы определили все используемые константы, но система справки все еще не знает, какие действия ей следует предпринимать, поскольку указанные темы отсутствуют в ее файле адресации. Добавим соответствующие строки в файл ShowStringX.hm.

```
HIDD_OPTIONS_STRING 0x02
HIDD_OPTIONS_BLACK 0x03
HIDD_OPTIONS_RED 0x04
HIDD_OPTIONS_GREEN 0x05
HIDD_OPTIONS_HORIZCENTER 0x06
HIDD_OPTIONS_VERTCENTER 0x07
HIDD_OPTIONS_OK 0x08
HIDD_OPTIONS_CANCEL 0x09
```

Убедитесь, что вы указали именно те числа, которые использовались в соответствующих директивах #define в файле OptionsDialog.h. Теперь все подготовительные работы проведены, осталось только написать текст самих функций. Вот как должна выглядеть функция OnHelpInfo().

```
BOOL COptionsDialog::OnHelpInfo(HELPINFO *lpHelpInfo)
{
  if (lpHelpInfo->iContextType == HELPINFO_WINDOW) // Должно присутствовать
                                // для контроля.
  {

    // Необходимо вызывать SDK WinHelp, а не CWinApp::WinHelp,
    // поскольку CWinApp::WinHelp не может принимать в качестве
    // параметра дескриптор.
    ::WinHelp((HWND)lpHelpInfo->hItemHandle,
      AfxGetApp()->m_pszHelpFilePath,
      HELP_WM_HELP, (DWORD)aHelpIDs);
  }
  return TRUE;
}
```

Эта функция просто вызывает функцию SDK WinHelp и передает ей для обработки дескриптор, путь к файлу справки, команду HELP_WM_HELP как требование вывести всплывающее окно с темой контекстно-зависимой справки и адрес созданной ранее таблицы, связывающей идентификаторы ресурсов с идентификаторами тем справки. От нашей функции, кроме вызова WinHelp(), никаких других действий не требуется.



Если вы ранее не встречались с подобной нотацией, когда терминальный оператор области действия (::) используется без указания перед ним имени класса, то это означает вызов функции, которая не является членом ни одного класса. При программировании для Windows это, как правило, означает обращение к функции SDK.

На заметку

Третий аргумент данного вызова функции WinHelp указывает системе справки на необходимость использовать определенный стиль окна справки. Значение HELP_WM_HELP указывает на требование вывести всплывающее меню так же, как и значение HELP_WM_CONTEXTMENU. Значение HELP_CONTEXT указывает на требование вывода обычного окна справки, которое может перемещаться, изменять размеры и имеет кнопки навигации в справочных текстах. Значение HELP_FINDER вызовет отображение диалогового окна Help Topics. Значения HELP_CONTENTS и HELP_INDEX являются устаревшими и должны быть заменены значением HELP_FINDER, если вы работаете с программой, в которой эти значения использовались ранее.

Функция OnContextMenu() выглядит еще проще.

```
void COptionsDialog::OnContextMenu(CWnd *pWnd, CPoint /*point*/)
{
  ::WinHelp((HWND)*pWnd, AfxGetApp()->m_pszHelpFilePath,
    HELP_CONTEXTMENU, (DWORD)aHelpIDs);
}
```

В этой функции нет необходимости проверять выполнение щелчка правой кнопкой мыши, как в функции OnHelpInfo(), поэтому она просто вызывает SDK WinHelp. Функция WinHelp() сама побеспокоится о выводе на экран контекстного меню с единственной командой What's This, а затем откроет окно справки для указанного объекта.

Проверьте введенные вами изменения, для чего оттранслируйте проект, выбрав команду Build⇨Build, а затем откомпилируйте файл справки, выделив файл ShowString.hpj и выбрав команду Build⇨Compile. (Другой вариант — сделать щелчок правой кнопкой мыши на имени файла ShowString.hpj на вкладке FileVeiw окна Project Workspace Window, а затем в раскрывшемся контекстном меню выбрать команду Compile.) Нет необходимости тестировать измененное приложение, поскольку та часть, которая генерируется с помощью мастера AppWizard, безусловно, будет работать, а без текстов справочных тем вновь добавленные вами в программу элементы не смогут ничего отобразить на экране.

Подготовка справочных текстов

Нужно подготовить тексты справочных тем в файлах формата .RTF с использованием специальных форматирующих кодов, которые будут означать нечто совершенно отличное от того, чем они обычно являются. Традиционно это принято делать в приложении Word, но в последнее время появилось достаточно много различных специализированных инструментов для подготовки текстов справок, которые значительно проще в использовании, чем Word. Однако в этом разделе вместо альтернативных инструментов мы рассмотрим методику подготовки справочных текстов с помощью Word. Тем не менее не забывайте, что существуют и более простые варианты и при разработке проектов достаточно солидных размеров вы сможете сохранить время и средства, если приобретете специализированный инструмент для подготовки справочных текстов. В книге *Designing Windows 95 Help* (которая упоминалась выше) целая глава посвящена выбору специализированных инструментов.



Открыть документ Word можно непосредственно из Visual Studio. Выберите команду File⇨Open и в открывшемся окне выберите требуемый файл. На экран будут выведены меню и панели инструментов Word. Это возможно по той причине, что документы Word являются объектами ActiveX Document, обсуждаемыми в главе 15. Большинство разработчиков предпочитают переключаться из Word в Visual Studio с помощью панели задач, вместо того, чтобы открыть в Visual Studio сразу нескольких файлов, и переключаться между этими документами с помощью меню Window. По этой причине описания операций в этом разделе предполагают, что вы запустите Word отдельно. Если вы предпочитаете работать исключительно в Visual Studio, ничто не мешает вам поступать именно так.

На рис. 11.6 показан файл afxcore.rtf, открытый в Word. Выберите команду View⇨Footnotes (Вид⇨Сноски) для отображения в нижней части экрана сносок — они имеются в файле. На рис. 11.6 показано, как выглядит текст, связанный с одним из идентификаторов темы справки. Выберите команду Tools⇨Options (Сервис⇨Параметры), в открывшемся диалоговом окне выберите вкладку View (Вид) и убедитесь, что флажок опции Hidden text (Скрытый текст) установлен. Скрытым текстом вводятся связи между темами справки. Каждая тема отделяется от следующей разделителем страниц (Page Break).

Существует восемь типов сносок, и каждый из них имеет свое назначение. Чаще всего из приведенного ниже списка используются только три первых типа сносок.

- **#, идентификатор темы справки.** Функция SDK WinHelp ищет этот идентификатор темы при выводе окна справки.
- **$, заголовок темы.** Этот заголовок выводится на экран по завершении поиска.
- **K, ключевые слова.** Эти слова выводятся на вкладку Index диалогового окна Help Topics.
- **A, A-ключевое слово.** Эти ключевые слова могут вызвать переход к соответствующей теме, но они не выводятся на вкладку Index диалогового окна Help Topics.
- **+, код просмотра.** Этот элемент определяет место данной темы в последовательности тем.
- **!, вход макроса.** Этот элемент определяет макрос темы, запускаемый, когда пользователь запросит вывод данной темы на экран.
- ***, тег построения.** Этот элемент используется для включения определенных тегов специального формата в файл справочной темы.
- **>, тип окна.** Данный элемент указывает тип окна, который следует использовать для этой темы.

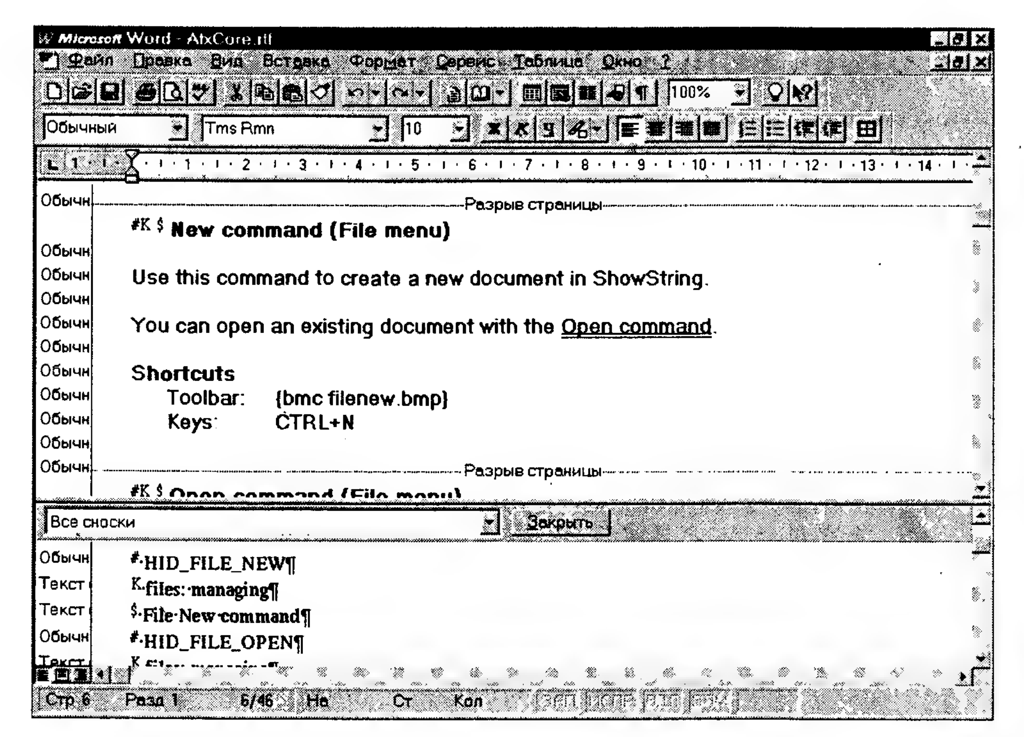


Рис. 11.6. Справочная тема, подготовленная мастером AppWizard по одной из стандартных команд меню, может быть отредактирована при помощи Word for Windows

Подчеркнутый двойной линией текст, за которым следует скрытый текст, указывает на возможность перехода к другой теме справки. Если пользователь сделает щелчок на этом элементе для перехода к другой теме, данная тема справки будет с экрана удалеиа. Если текст перед скрытым текстом подчеркнут одной линией, при обработке связи будет раскрыто всплывающее окно поверх окна с даиной темой справки, в котором содержится определение или краткое замечание. (Вам могут встретиться и такие файлы со справочными текстами, в которых присутствует перечеркнутый текст; это в точности то же самое, что и подчеркивание двойной линией, т.е. переход к другой справочной теме.) Во всех этих случаях скрытый текст содержит идентификатор темы справки, на которую будет сделан переход или которая будет помещена во всплывающее окно.

На рис. 11.7 показано, какой справочный текст для команды File⇨New будет выведен в приложении ShowString. Чтобы увидеть его на своем дисплее, запустите приложение ShowString, для чего в окне Visual Studio выберите команду Build⇨Execute. В окне приложения ShowString выберите команду Help⇨Help Topics. Откройте книгу меню и сделайте двойной щелчок на теме, касающейся меню File, а затем щелкните на команде New.

Настал момент отложить на время программирование и подготовить для приложения ShowString все справочные темы, перечисленные в приведенном выше списке. (Его вы найдете выше в этой же главе, в разделе *Планирование структуры справочной системы*.) Ниже изложение построено в расчете, что вы работаете с Word.

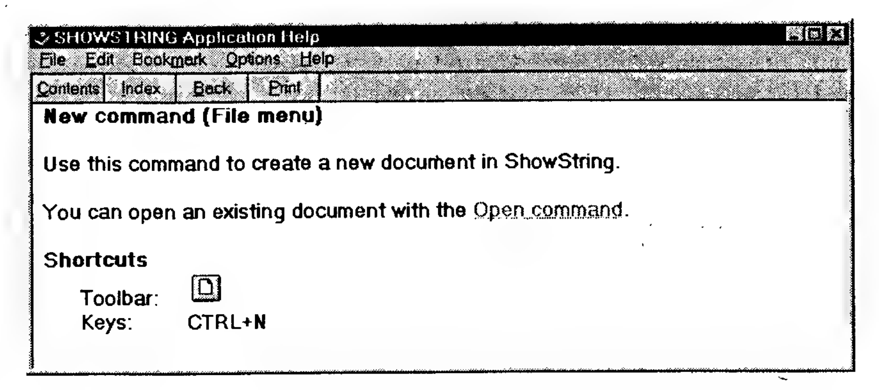


Рис. 11.7. В окне приложения ShowString выведена справочная информация по одной из тем, сгенерированная AppWizard

Замена строки-местодержателя

Для изменения строки-местодержателя, помещенной мастером AppWizard в созданные им файлы справочных тем, откройте в Word файл afxcore.rtf. (Этот файл расположен в папке HLP, находящейся в папке проекта ShowString.) Затем выполните следующие операции.

1. Поместите курсор в самое начало документа и выберите команду Edit⇨Replace (Правка⇨Заменить).
2. Введите в поле Find What (Что искать) значение <<YourApp>>, а в поле Replace With (Заменить на) — значение ShowString.
3. Щелкните на кнопке Replace All (Заменить все).

Откройте файл afxprint.rtf и повторите эти операции.

Вернитесь в файл afxcore.rtf и выполните поиск в тексте символов << (для этой цели используйте команду Edit⇨Find (Правка⇨Найти) и не забудьте, что комбинацию клавиш <Shift+F4> можно использовать для повторения предыдущего действия). Эти символы указывают, куда следует внести изменения или где принять решение. Для приложения ShowString в файле afxcore.rtf необходимы следующие изменения.

1. Первой секцией в этом файле является индекс справки приложения ShowString. Удалите секцию How To и сопутствующие комментарии. В реальном приложении в этом месте вам надо будет добавить новые темы.
2. В следующей секции, после разделителя страниц, находится таблица, описывающая команды меню File. Поскольку в меню нашего приложения нет команды Send, удалите строку этой команды из таблицы меню File.
3. Третья секция является таблицей, описывающей команды меню Edit. Удалите строки Paste Link, Insert New Object и Links.
4. Четвертая секция используется для меню View и не требует внесения изменений.
5. Пятая секция используется для меню Window. Удалите строку Split из таблицы этого меню.
6. Шестая секция используется для меню Help и не требует внесения изменений.
7. Седьмая секция содержит данные о команде File⇨New. Удалите предложение о выборе типа файла и напоминание о необходимости его удаления.

8. Целиком удалите восьмую секцию, содержащую текст, который выводится в диалоговое окно при описании команды File⇨New. Удалите и один разделитель страниц (но не оба сразу). Если вы полностью удаляете какую-либо секцию, удалите и один из разделителей страниц так, чтобы файл не содержал двух разделителей страниц, стоящих рядом.

9. Следующая секция описывает команду File⇨Open и не требует внесения изменений.

10. Перейдите к тексту, выводимому в диалоговое окно при описании команды File⇨Open. Отредактируйте текст так, чтобы в нем указывалось, что список файлов в поле Type содержит единственное значение All Files.

11. Пролистайте файл до тех пор, пока не найдете тему о команде File⇨Send. Полностью удалите ее, включая и стоящий в ее конце разделитель страниц.

12. В описании команды File⇨Save As удалите предложение вывести описание прочих режимов, поскольку их нет.

13. Когда вы дойдете до описания команды Edit⇨Undo, то сможете лучше понять, почему программы, написанные после того, как были подготовлены по ним руководства, работают лучше. В приложении ShowString, построенном по методу, описанному в главе 8, команда Undo всегда будет неактивна, как и команды Cut, Copy и Paste. Можно удалить темы, посвященные этим неподдерживаемым в приложении командам, но, вероятно, лучше будет подготовить и добавить поддержку этих команд в следующей версии данного приложения. Добавьте в файл некоторый текст, поясняющий, что в данной версии приложения эти команды не реализованы. Сохраните раздел, описывающий комбинации клавиш для этих команд, чтобы пользователь имел возможность убедиться, что при вводе комбинации клавиш <Ctrl+Z> никакие действия не выполняются.

14. Пролистайте файл до темы Toolbar, где вы встретите следующее напоминание: << Add or remove toolbar buttons from the list below according to which ones your application offers. >> (Добавьте в расположенный ниже список пиктограммы панели инструментов или удалите их в соответствии с тем, что поддерживается в вашем приложении.) Удалите это напоминание и ссылки на пиктограммы Undo (Откат), First Record (Первая запись), Previous Record (Предыдущая запись), Next Record (Следующая запись) и Last Record (Последняя запись).

15. Примерно в середине оставшейся части файла расположена тема по команде Window⇨Split. Удалите всю тему.

16. Пролистайте файл до темы по команде Help⇨Index и удалите ее. Также удалите темы по командам Help⇨Using Help и Help⇨About.

17. В теме справки по Title Bar (строка заголовка) удалите директиву о вставке графики. Если вы предпочитаете следовать указаниям этой директивы, создайте графический файл типа .bmp с изображением панели заголовка с помощью инструментов снятия копии экрана. Обрежьте изображение, оставив на нем только панель заголовка, и вставьте графику вместе с директивой bmc так, как это сделано несколькими строками ниже для графического элемента bullet.bmp.

18. Поскольку представление ShowString порождено не от класса CScrollView, оно не поддерживает прокрутки. Удалите тему Scrollbars вместе с ее разделителем страниц.

19. В теме по команде Close (но не по команде File⇨Close, тема которой расположена в файле значительно выше этого места) описание комбинации клавиш <Alt+F4> должно выглядеть так: закрывает приложение ShowString.

20. Удалите описание диалоговых окон Ruler, Choose Font, Choose Color, Edit Find, Find Dialog, Edit Replace, Replace, а также темы по командам Repeat, Clear, Clear All (все из меню Edit), Next Pane и Previous Pane.

21. Пропустите тему How To Modify Text целиком, оставив ее без изменения.

22. Удалите последнюю директиву о необходимости добавления сообщения No Help Available (Справочные данные отсутствуют) из каждого окна сообщения.

На этом значительные изменения в автоматически сформированном мастером AppWizard файле afxcore.rtf можно считать выполненными. Другой подобный файл afxprint.rtf просто пролистайте до самого конца и удалите тему Page Setup.

Не хотите ли проверить, как все это теперь будет работать? Сохраните в Word файлы afxcore.rtf и afxprint.rtf. Переключитесь в Visual Studio и выберите команду Build⇨Build с целью обновления состояния проекта. Затем откройте файл ShowString.hpj и выберите команду Build⇨Compile. Эта команда необходима для объединения в файле ShowString.hlp всех файлов .rtf. Для запуска приложения ShowString выберите команду Build⇨Execute. В окне приложения ShowString выберите команду Help⇨Help Topics. Как видно из рис. 11.8, размер справочной темы Window menu теперь существенно уменьшился. Точно так можно проверить результат внесения вами всех остальных изменений.

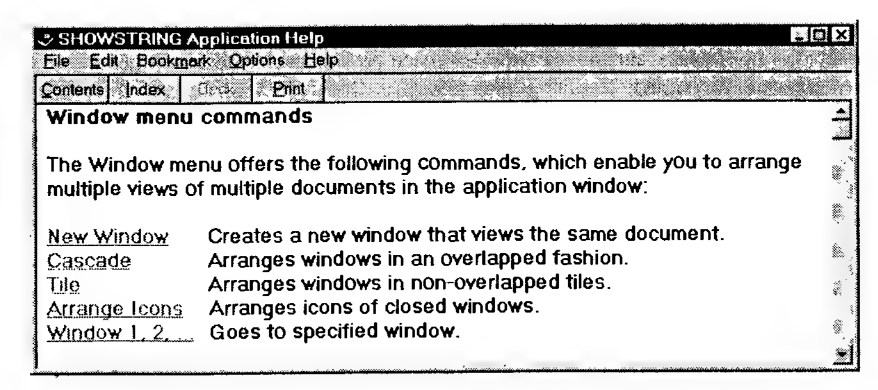


Рис. 11.8. Протестировать внесенные изменения можно после сохранения файлов .rtf и перекомпиляции проекта справочной системы

Добавление новых тем

При добавлении новых тем не следует вносить их в файлы, автоматически подготовленные мастером AppWizard. Эти файлы следует сохранять неизменными до тех пор, пока вам, к примеру, не потребуется изменить описание команды File⇨Open и т.п. Для новых тем создайте новый файл, выбрав в Word команду File⇨New (Файл⇨Создать) и сохранив его под именем ShowString.rtf в папке HLP, имеющейся в папке проекта ShowString. (Не забудьте при сохранении файла установить в поле Save As File Type (Тип файла) тип Rich Text Format.) Если вы работаете с крупным проектом, можете подготовить несколько отдельных файлов .rtf, а в нашем случае достаточно будет одного файла. В Visual Studio откройте файл ShowString.hpj, сделав на вкладке FileView двойной щелчок на его имени, и найдите в нем секцию с заголовком [FILES]. Добавьте в конец этой секции следующую строку:

```
showstring.rtf
```

Меню Tools

Вернитесь в Word, переключитесь в файл afxcore.rtf и скопируйте в буфер обмена тему, посвященную меню File. Вернитесь в файл ShowString.rtf и вставьте в него содержимое буфера обмена. (Не забудьте при копировании включить в выделенный фрагмент и разделитель

страницы в конце темы.) Для вывода на экран сносок выберите команду View⇨Footnotes (Вид⇨Сноски). Выберите команду Tools⇨Options (Сервис⇨Параметры) и на вкладке View (Вид) установите флажок опции Hidden Text (Скрытый текст) для отображения на экране скрытого текста документа. Теперь следует отредактировать скопированную тему о меню File и превратить ее в тему меню Tools. Прежде всего изменим сноски. В результате они приобретут следующий вид.

- Сноска # содержит идентификатор темы справки. Система Help использует его для поиска этой темы при переходе со страницы оглавления. Измените ее значение на menu_tools.
- Сноска K является входом по ключевым словам. Хотя диалоговое окно Options нашего приложения вполне можно охарактеризовать несколькими ключевыми словами, для данного меню это излишне. Поэтому удалите сноску, выбрав букву K в тексте темы справки и нажав клавишу <Del>. Нужно именно выбрать данную букву, а не просто сделать перед ней щелчок. Сноска будет автоматически удалена вместе с буквой.
- Сноска $ определяет заголовок темы. Измените ее значение на Tools menu commands (Команды меню Tools).

В тексте справочной темы замените в первых двух строках слово File словом Tools и удалите все строки таблицы, кроме одной. Замените подчеркнутый текст этой строки словом Options, скрытый текст этой строки замените текстом HID_TOOLS_OPTIONS, а в поле правого столбца в этой строке введите значение Changes string, color, and centering (Изменяет текст строки, ее цвет и центрирование). На рис. 11.9 показано, как должен выглядеть файл ShowString.rtf в окне Word после внесения всех этих изменений.

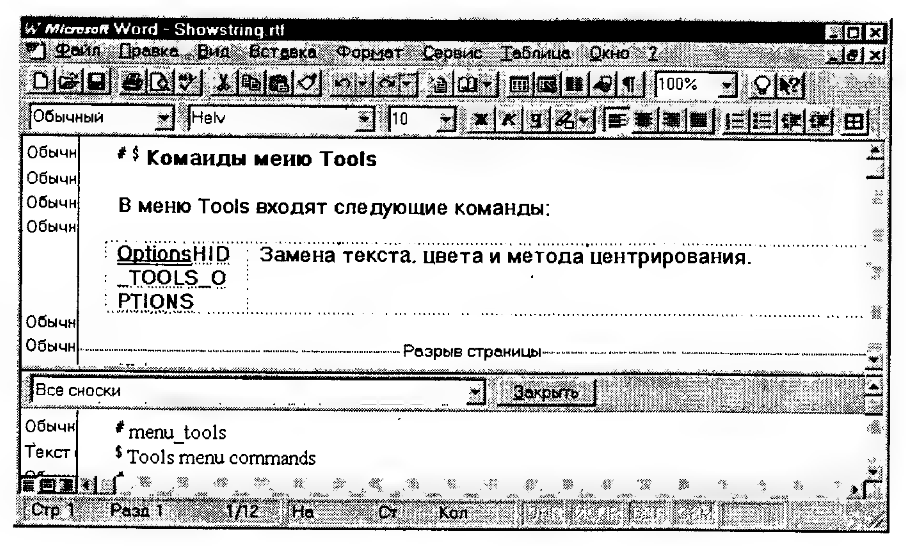


Рис. 11.9. Внесите в файл ShowString.rtf справку о новой команде меню



Если вы не помните идентификаторы тем справки, используемые в ввшем приложении, обратитесь к файлам .hm. Те идентификаторы, которые были добавлены с помощью Visual Studio, например HID_TOOLS_OPTIONS для команды меню, находятся в файле ShowString.hm. В файле же ShowStringx.hm находятся идентификаторы, вручную добавленные для контекстной справки.

Команда Tools⇨Options

Вновь перейдите в файл afxcore.rtf и скопируйте в буфер обмена справочную тему о команде File⇨New, а затем вставьте ее в конец файла ShowString.rtf. Тема и относящиеся к ней сноски копируются совместно. Тщательно просмотрите сноски и удостоверьтесь, что вы работаете с теми, которые относятся к теме команды Options меню Tools, а не к теме самого меню Tools. Выполните следующие действия.

1. Измените значение сноски # на HID_TOOLS_OPTIONS.
2. Измените ключевые слова в сноске K. Здесь нам потребуется ввести несколько ключевых слов, причем одно значение от другого следует отделять точкой с запятой (;). Некоторые из этих ключевых слов будут двухуровневыми, причем уровни будут разделяться запятыми. На первый случай введем в сноску такие значения: string, changing;color, changing;centering, changing; appearence, controlling.
3. Измените значение сноски $ на Tools Options command (Команда Tools⇨Options).
4. Замените содержимое первой строки темы справки значением Options command (Tools menu) (Команда Options меню Tools).
5. Удалите остальную часть темы справки и замените ее кратким описанием данной команды меню. Весьма подойдет следующий текст:

Используйте эту команду для изменения формата окна приложения с помощью диалогового окна Options. Текст сообщения, цвет, центрирование – все эти параметры настраиваются в диалоговом окне.

Если у вас возникло желание протестировать внесенные изменения, сохраните все файлы, открытые в Word, откомпилируйте проект справочной системы и запустите приложение ShowString. В окне этого приложения откройте меню Tools, выделите команду Options, соответственно установив полосу выбора с помощью клавиш управления курсором (вместо простого щелчка на этой команде), и нажмите клавишу <F1>. На рис. 11.10 показано окно справки, которое должно появиться на экране в результате всех этих манипуляций.

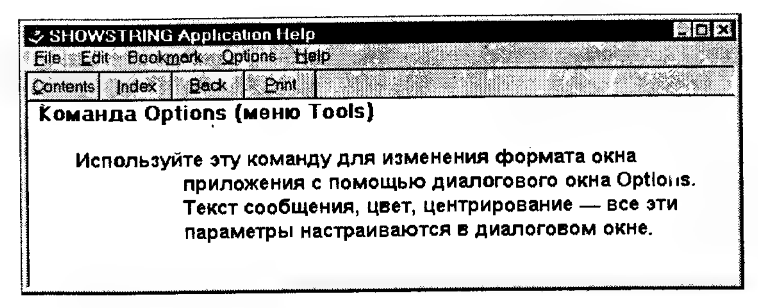


Рис. 11.10. Новое окно справки о команде Tools⇨Options

Элементы управления в диалоговом окне Options

Еще раз скопируйте в файл ShowString.rtf тему о команде New из меню File и удалите из нее все ненужное. Сделать это можно, выполнив следующие операции.

1. Удалите сноски K и $.
2. Измените значение сноски # на HIDD_OPTIONS.
3. Измените значение в первой строке на Options dialog (Диалоговое окно Options).
4. Удалите весь остальной текст темы.

Скопируйте этот блок текста в буфер обмена и вставьте его в файл еще семь раз, создав таким образом заготовки для описания всех элементов управления в данном диалоговом окне. Кроме того, не забудьте скопировать для каждого элемента разделитель страниц в начале или конце текста. Далее отредактируйте каждую из заготовок, превратив их в справочные темы со следующими идентификаторами тем.

- HIDD_OPTIDNS_STRING
- HIDD_OPTIONS_BLACK
- HIDD_OPTIONS_RED
- HIDD_DPTIONS_GREEN
- HIDD_DPTIONS_HORIZCENTER
- HIDD_DPTIONS_VERTCENTER
- HIDD_OPTIONS_DK
- HIDD_OPTIONS_CANCEL

Замените идентификаторы тем и добавьте одно-два предложения описательного характера. Будьте кратки. В файлах примера для этой главы все темы состоят из одного предложения, которое начинается с глагола в повелительном наклонении, например, *Щелкните* или *Выберите*, и заканчивается точкой (.). Если вы отдадите предпочтение иному стилю построения сообщений во всплывающих окнах, то используйте один и тот же стиль для всех сообщений. Непоследовательность в стиле сообщений приводит пользователя в замешательство и вызывает у него недоверие к качеству ваших программ в целом.

Для проверки результата выполненной работы еще раз откомпилируйте файл ShowString.hpj, запустите приложение ShowString и выберите команду Tools⇨ Options. Щелкните на пиктограмме с вопросительным знаком, а затем щелкните где-либо в диалоговом окне. Проверьте сообщения, выводимые для каждого из элементов управления, и убедитесь, что введенный вами справочный текст корректен. На рис. 11.11 показано окно приложения ShowString, в котором выведена контекстная справка о поле ввода String.

Справка относительно пункта Understanding Centering

Поместите в файл ShowString.rtf очередную копию справочной темы о команде New в меню File. Внесите в нее следующие изменения.

1. Измените значение сноски # на HID_CENTERING (этот идентификатор темы вы внесли в файл ShowString.hm, он же вызывается в методе CShowString::OnHelpUnderstandingcentering()).
2. Измените значение сноски K на Centering (Центрирование).
3. Измените значение сноски $ на Understanding Centering (Понятие о центрировании).

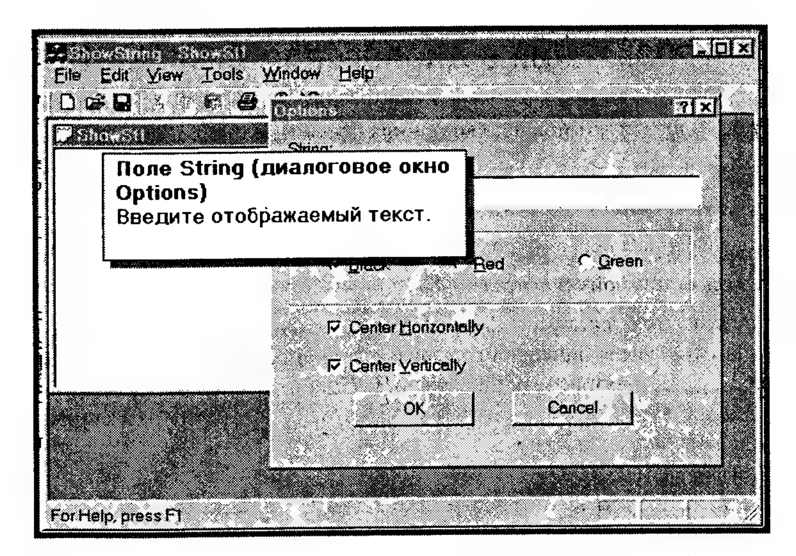


Рис. 11.11. Контекстная справка об элементе управления в диалоговом окне

4. Замените текст заголовка в первой строке текстом Понятие о центрировании.
5. Замените текст темы кратким объяснением понятия центрирования, например таким:

```
Приложение ShowString может выполнить центрирование отображаемой строки. В
диалоговом окне Options, которое выводится на экран при выборе команды Options в
меню Tools, можно независимо установить два режима центрирования - горизонталь-
ное и вертикальное. Текст, не имеющий горизонтального центрирования, отобража-
ется с левого края окна. Текст, не имеющий вертикального центрирования, отобра-
жается в верхней части окна.
```

6. Добавьте в этот текст связи, которые идут от слова Tools к справочной теме menu_tools и от слова Options к теме HID_TOOLS_OPTIONS, определенной выше. Не забывайте о дополнительных пробелах.

Подготовьте приложение к тестированию внесенных изменений описанным выше способом и запустите его. В окне приложения выберите команду Help⇨Understanding Centering — и увидите на экране то, что изображено на рис. 11.12. Проверьте, как работают перекрестные связи. Для возврата к теме о центрировании можно использовать кнопку Back.

Изменение справочной темы Как модифицировать текст

Мастер AppWizard предоставляет готовую справочную тему How to Modify Text (Как модифицировать текст) в самом конце файла afxcore.rtf. Ее следует откорректировать так, чтобы она содержала разъяснения о том, как работает приложение ShowString. Эта тема отображается, когда пользователь при запросе контекстной справки делает щелчок в поле окна. Замените существующий текст значительно более кратким объяснением, рекомендующим пользователю выбрать команду Tools⇨Options. Для добавления к этой теме связи введите HID_TOOLS_OPTIONS сразу после находящегося в тексте справки слова Options. Здесь же рядом введите menu_tools сразу после слова Tools. Выберите слово Options и нажмите клавиши <Ctrl+Shift+D>, чтобы подчеркнуть его двойной линией. Повторите эту операцию со словом Tools. Выберите слово

HID_TOOLS_OPTIONS и нажмите клавиши <Ctrl+Shift+H> для превращения его в скрытый текст. Повторите эту операцию со словом menu_tools.

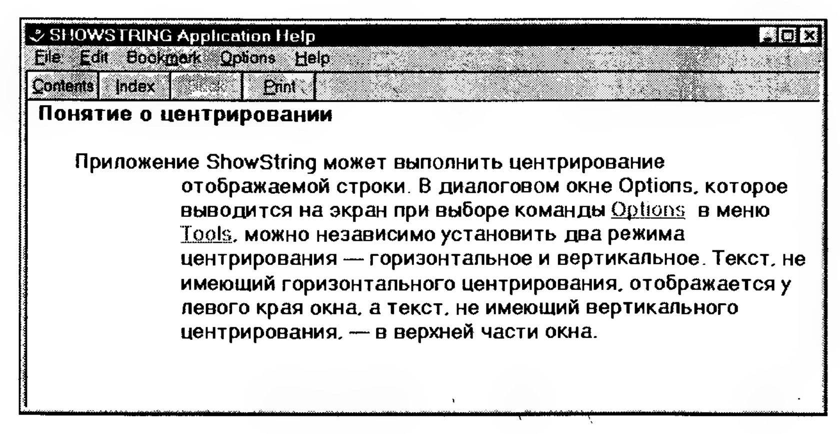


Рис. 11.12. Отображение на экране справочной темы после ее выбора в меню Help



Если в Word данные комбинации клавиш переопределены, необходимое форматирование можно выполнить и другими методами. Для подчеркивания текста двойной линией выделите его и выберите команду Format⇨Font (Формат⇨Шрифт). Раскройте список Underline (Подчеркивание) и выберите значение Double (Двойное), а затем щелкните на кнопке OK. Для превращения обычного текста в скрытый выделите его и выберите команду Format⇨Font, установите флажок опции Hidden (Скрытый) и щелкните на кнопке OK.



Между текстом, подчеркнутым двойной линией, и скрытым текстом не должно быть пробелов, как и в конце скрытого текста. По этому поводу Word может причинить вам некоторое беспокойство, поскольку режим Smart Cut and Paste (Учет пробелов при вырезании и вставке), позволяющий так изящно перетаскивать слова с помощью мыши, автоматически вставляет дополнительные пробелы там, где вы этого не ожидаете, а режим Automatic Word Selection (Автоматическое выделение слов) делает невозможным выделение только части слова. Можно отключить эти режимы, выбрав команду Tools⇨Options (Сервис⇨Параметры) и вкладку Edit (Правка), а затем сбросив флажки опций Automatic Word Selection (Автоматически выделять слова) и Use Smart Cut and Paste (Учитывать пробелы при вырезании и вставке).

Вы готовы снова протестировать полученные результаты? Сохраните файлы в Word, откомпилируйте файл проекта системы справки и запустите приложение ShowString. В окне приложения щелкните на пиктограмме What's This панели инструментов, а затем щелкните в поле окна. Должно появиться окно с вновь добавленной нами темой *How to Modify Text*.

Реорганизация оглавления справочной системы

Теперь наше совершенно крошечное приложение практически полностью документировано. Осталось добавить в оглавление справочной системы темы о меню Tools и Understanding Centering и реорганизовать индекс. Проще всего работать с оглавлением с помощью программы Help Workshop. Закройте все связанные со справочной системой файлы, ко-

торые были вами открыты в Visual Studio и Word, и запустите приложение Help Workshop (для этого в списке Programs нужно выбрать Microsoft Visual Studio 6.0⇨ Microsoft Visual Studio 6.0⇨Tools⇨Help Workshop). Откройте файл ShowString.cnt, выбрав команду File⇨Open и осуществив поиск этого файла в диалоговом окне Open. Данный файл является файлом оглавления справочной системы приложения ShowString.

В первой открытой книге щелкните на элементе View Menu, а затем щелкните на кнопке Add Below (Вставить под). (Альтернативный вариант — щелкнуть на элементе Window Menu, а затем щелкнуть на кнопке Add Above (Вставить перед).) Раскроется диалоговое окно Edit Contents Tab Entry, показанное на рис. 11.13. Заполните его поля так, как это показано на рисунке. Если последние два поля оставлены вами пустыми, по умолчанию для них принимаются значения Help File и Window Type. Щелкните на кнопке OK.

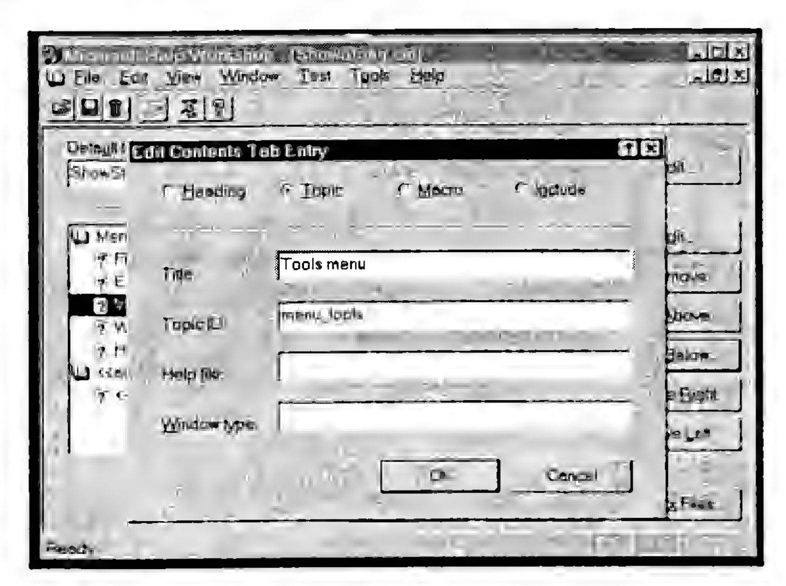


Рис. 11.13. В диалоговом окне Edit Contents Tab Entry *приложения* Help Workshop *добавьте новые пункты в оглавление справочной системы*

Щелкните на книге-местодержателе, помеченной надписью <<add your application-specific topics here>>, а затем еще раз щелкните на кнопке Add Above. Во вновь раскрывшемся диалоговом окне Edit Contents Tab Entry установите переключатель Heading в верхней части окна. В данном случае, как это видно на рис. 11.14, ввод допускается только в поле заголовка. Не следует вводить значение Understanding Centering, поскольку оно же является заглавием единственной темы, помещаемой под данный заголовок. Щелкните на кнопке OK.

Добавьте под новый заголовок тему Understanding Centering, имеющую идентификатор HID_CENTERING, и удалите заголовок и тему местодержателя. Сохраните введенные изменения, закройте приложение Help Workshop, еще раз в Visual Studio откомпилируйте файл ShowString.hpj и протестируйте работу справочной системы приложения ShowString. Выберите в его окне команду Help⇨Help Topics, и вы увидите на экране то, что изображено на рис. 11.15.

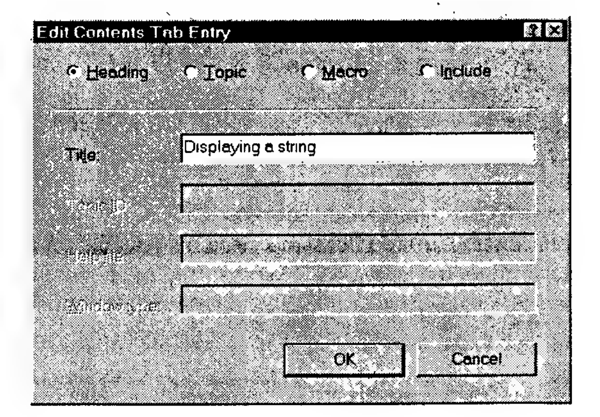


Рис. 11.14. Заголовки на вкладку оглавления справочной системы добавляются в диалоговом окне Edit Contents Tab Entry *приложения* Help Workshop *после установки переключателя* Heading

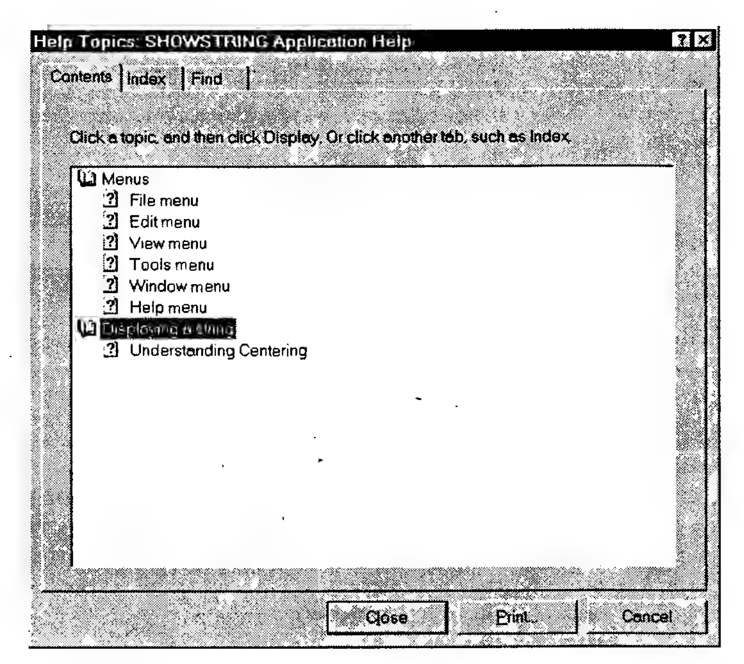


Рис. 11.15. Отредактированное оглавление справочной системы

В открытом диалоговом окне Help Topics щелкните на корешке вкладки Index. На рис. 11.16 показано, как будут выглядеть все введенные нами во время работы над данным разделом сноски типа K, сведенные в единый индекс. Если он покажется вам несколько бедноватым, вы всегда имеете возможность обратиться к файлам `.rtf` и добавить в них произвольное число новых ключевых слов, не забывая разделять их точкой с запятой.

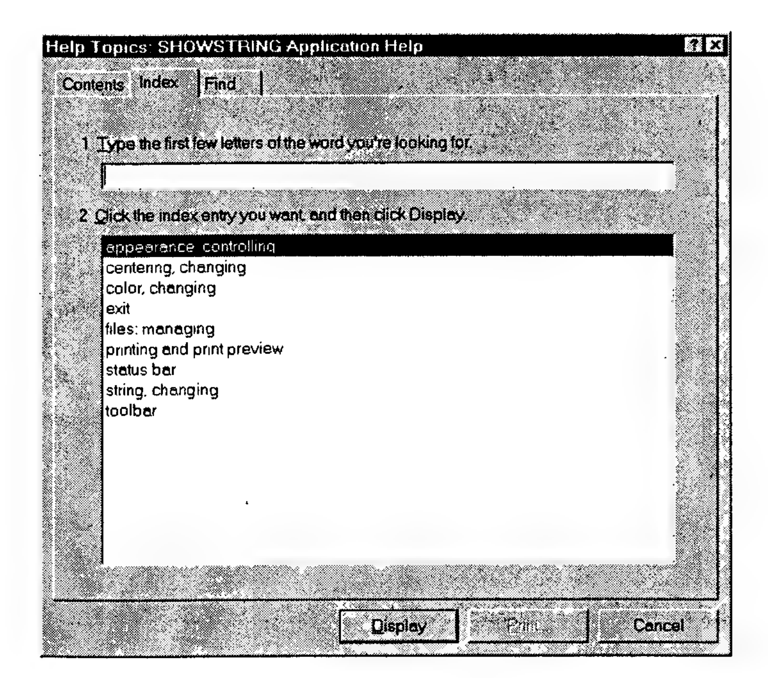


Рис. 11.16. Индекс строится из содержимого сносок типа K, помещенных в файлы `.rtf`

ГЛАВА 12

Вкладки и окна свойств

В этой главе...

Знакомство с окнами свойств

Создание приложения Property Sheet Demo

Запуск приложения Property Sheet Demo

Добавление окон свойств к приложениям

Преобразование окна свойств в мастер

Знакомство с окнами свойств

Одним из новейших типов графических объектов являются диалоговые окна, снабженные вкладками, которые называются *окнами свойств* (property sheet). Окно свойств представляет собой диалоговое окно, имеющее две или более вкладки. Windows 95 работает с множеством окон свойств, каждое из которых объединяет несколько опций, предоставляя пользователю возможность устанавливать и модифицировать их. Переключение между вкладками в окне свойств выполняется с помощью щелчков мышью на расположенных в верхней части диалогового окна корешках вкладок, помеченных соответствующими этикетками. Используя подобные диалоговые окна для формирования сложных групп опций, Windows 95 предоставляет пользователю простой способ получить доступ к информации и выполнить требуемые установки. Вы, вероятно, уже догадались, что Visual C++ 6.0 поддерживает работу с окнами свойств Windows 95, используя для этой цели классы CPropertySheet и CPropertyPage.

Мастера (Wizard) схожи с окнами свойств, но в них для перемещения между страницами используются кнопки, а не корешки вкладок. Вам уже неоднократно приходилось иметь дело с мастерами. Эти специализированные типы диалоговых окон помогают пользователю шаг за шагом выполнять сложные операции. Например, когда для генерации исходного текста программ нового проекта вы используете AppWizard, этот мастер проводит вас через весь соответствующий процесс. Вы управляете мастером, щелкая на кнопках Back (Назад), Next (Далее) и Finish (Готово).

Найти пример окна свойств в Windows 95 так же просто, как найти песок в пустыне. Достаточно выбрать в любом подходящем меню команду Properties (Свойства) или вывести на экран диалоговое окно Options в большинстве приложений. Например, на рис. 12.1 показан пример диалогового окна, которое открывается по команде Tools⇒Options в Visual Studio. Это диалоговое окно содержит 12 вкладок свойств, в каждой из которых сгруппированы средства настройки определенной категории.

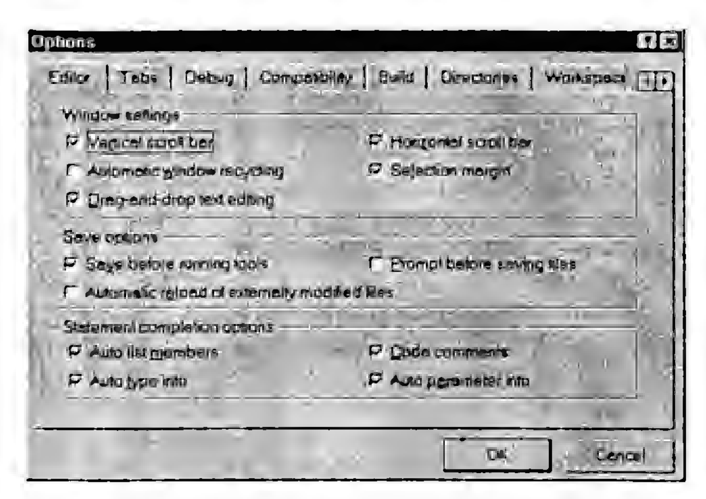


Рис. 12.1. Диалоговое окно свойств Options имеет множество вкладок для настройки среды разработки

На заметку Многие забывают о различиях, существующих между окном свойств (property sheet) и вкладкой окна свойств (property page). Окно свойств включает *вкладки свойств*. Вкладки окна свойств — это окна, которые содержат элементы управления. Они выводятся внутри окон свойств.

Из всего этого следует, что окна свойств представляют собой отличное средство организации множества связанных между собой опций. Прошли времена диалоговых окон, переполненных опциями до такой степени, что разобраться в них можно было только после специального курса обучения на уровне колледжа. В последующих разделах этой главы будут рассматриваться методы, которые вы сможете применить для создания собственных окон свойств, снабженных вкладками, на базе классов MFC CPropertySheet и CPropertyPage.

Создание приложения Property Sheet Demo

Теперь, получив общее представление об окнах свойств, мы приступим к изучению методики построения приложения, в котором будут использоваться эти удобные специализированные диалоговые окна. С этой целью будет создано демонстрационное приложение, в ходе разработки которого будет продемонстрирована технология создания и настройки окон свойств.

Создание основных файлов проекта

Прежде всего создайте основные файлы программы Property Sheet Demo с помощью AppWizard. Настройку мастера нужно выполнить соответственно приведенной ниже таблице. Для построения файлов проекта щелкните на кнопке OK в последнем диалоговом окне настройки AppWizard.

Имя диалогового окна	**Опции, которые следует установить**
Вкладка New Project	Присвойте новому проекту имя Propsheet и укажите путь к той папке, в которой будут сохраняться файлы проекта. Убедитесь, что выбрано значение MFC AppWizard (exe). Для всех остальных опций оставьте установки, предложенные мастером по умолчанию
MFC AppWizard Step 1	Установите переключатель Single document
MFC AppWizard Step 2 of 6	Сохраните параметры настройки, предложенные мастером по умолчанию
MFC AppWizard Step 3 of 6	Сохраните параметры настройки, предложенные мастером по умолчанию
MFC AppWizard Step 4 of 6	Сбросьте все флажки
MFC AppWizard Step 5 of 6	Сохраните параметры настройки, предложенные мастером по умолчанию
MFC AppWizard Step 6 of 6	Сохраните параметры настройки, предложенные мастером по умолчанию

После завершения работы на экран будет выведено информационное диалоговое окно New Project Information, показанное на рис. 12.2.

Редактирование ресурсов

Теперь можно приступить к редактированию ресурсов приложения, подготовленных AppWizard: удалению ненужных меню и пиктограмм панели инструментов, редактированию окна About и, самое главное, добавлению команды меню, по которой на экран будет выводиться окно свойств. Выполните следующие действия.

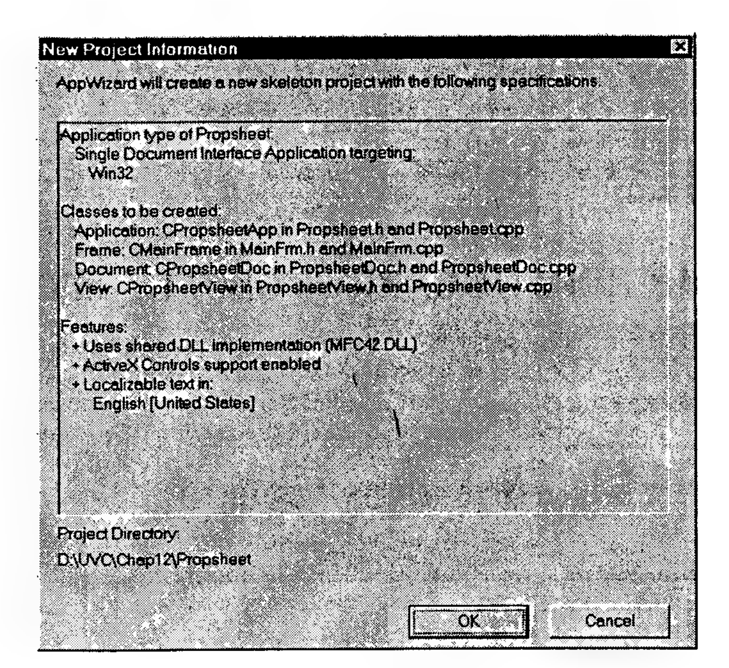


Рис. 12.2. Диалоговое окно New Project Information *после настройки* AppWizard

1. В окне разработки проекта в левой части экрана выберите вкладку ResourceView. В Visual Studio раскроется окно ResourceView, показанное на рис. 12.3.

2. Разверните в окне ResourceView дерево ресурсов приложения — щелкните на знаке "плюс", стоящем перед элементом `Propsheet resources`. Далее щелкните на знаке "плюс", расположенном перед элементом `Menu`, и сделайте двойной щелчок на идентификаторе меню `IDR_MAINFRAME`. В результате на экране появится окно редактора меню Visual C++, в котором будет отображено меню `IDR_MAINFRAME` приложения в том виде, как его сформировал мастер `AppWizard`.

3. Удалите из него пункт Edit — щелкните на этом пункте в окне редактора меню (но не на меню Edit Visual Studio), а затем нажмите клавишу <Del>. Появится диалоговое окно, запращивающее подтверждение команды удаления. Щелкните в нем на кнопке OK.

4. В меню Help сделайте двойной щелчок на команде About Propsheet. Раскроется диалоговое окно свойств этой команды. В поле Caption введите `&About Property Sheet Demo`. Зафиксируйте диалоговое окно свойств на экране в выбранной позиции, щелкнув на пиктограмме канцелярской кнопки в верхнем левом углу диалогового окна.

5. В меню File создаваемого приложения удалите все команды, кроме команды Exit.

6. Выберите в нижней части меню File пустой шаблон команды и введите в поле Caption его окна свойств значение `&Property Sheet...`, а в поле ID — идентификатор команды `ID_PROPSHEET`, как показано на рис. 12.4. Затем с помощью мыши так перетащите новую команду меню в позицию над Exit, чтобы она стала в меню File первой.

7. В окне ResourceView щелкните на знаке "плюс" перед элементом `Accelerator` и выделите значение идентификатора акселератора `IDR_MAINFRAME`. Нажмите клавишу <Del> и удалите все акселераторы из приложения.

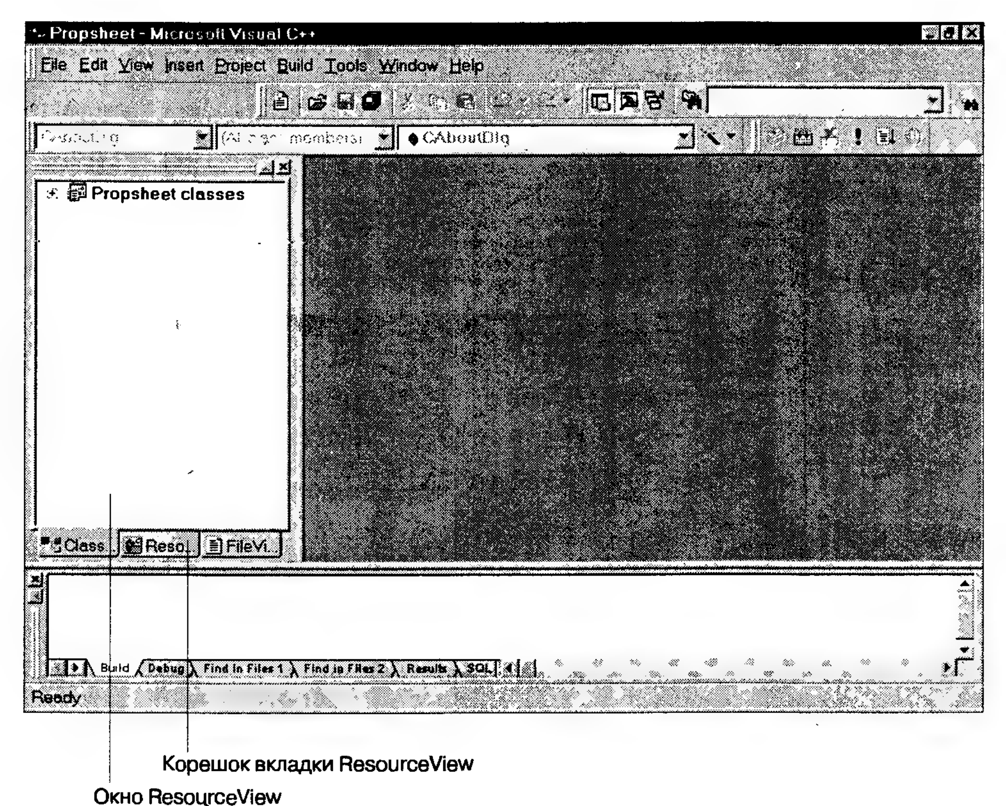


Рис. 12.3. Щелчок на корешке вкладки ResourceView вызывает на экран окно ResourceView

8. В окне ResourceView щелкните на знаке "плюс" перед элементом `Dialog Resource`. После двойного щелчка на идентификаторе диалогового окна `IDD_ABOUTBOX` на экран будет выведен редактор диалоговых окон.

9. Для вывода свойств редактируемого диалогового окна в целом щелкните на его строке заголовка. Введите в поле Caption новое значение заголовка: `About Property Sheet Demo`.

10. Щелкните на первом элементе типа "статический текст" и измените в поле Caption его значение на `Property Sheet Demo, Version 1.0`. Щелкните на втором элементе и добавьте в поле Caption текст `Que books` в конец существующего.

11. Добавьте третий элемент статического текста: `Special Edition Using Visual C++ 5`. Теперь окно About приложения должно принять вид, показанный на рис. 12.5. Закройте редактор диалоговых окон.

12. Щелкните на знаке "плюс" перед элементом `String Table` в окне ResourceView. После двойного щелчка на дочернем элементе `String Table` откроется окно редактора таблицы строк.

13. Сделайте двойной щелчок на строке `IDR_MAINFRAME`, а затем измените первый сегмент строки на `Property Sheet Demo`, как показано на рис. 12.6. Назначение этого ресурса будет обсуждаться ниже, в главе 16. Та надпись, которую вы только что изменили, появится в строке заголовка окна приложения.

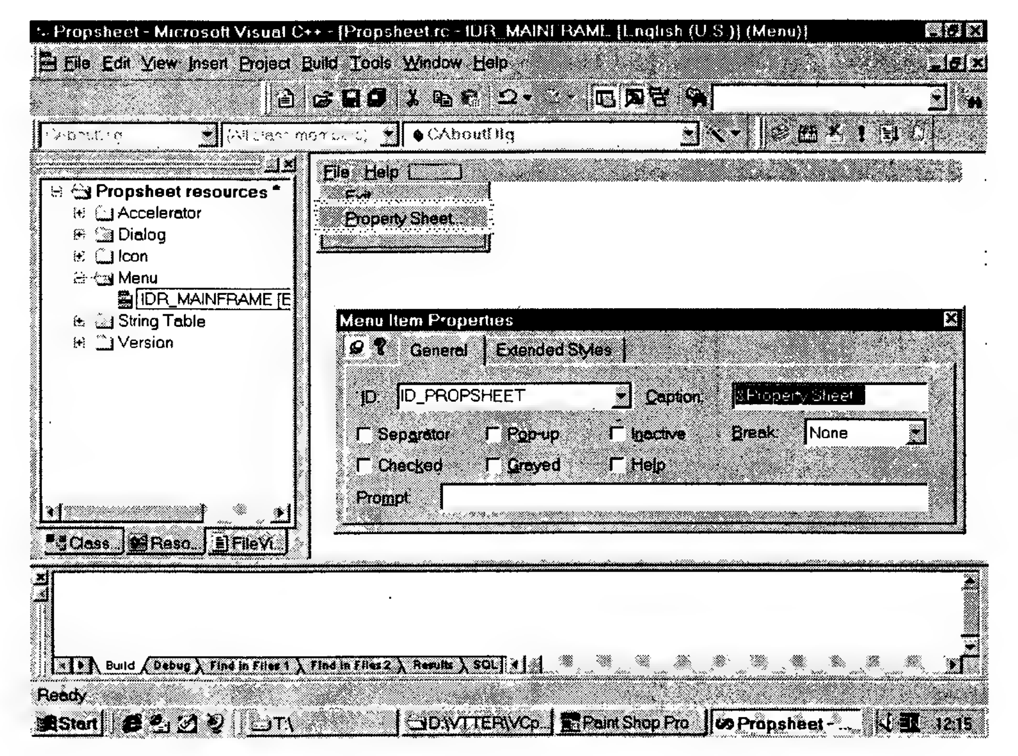


Рис. 12.4. Добавьте в меню File *команду* Property Sheet

Добавление новых ресурсов

Теперь, когда все основные ресурсы приложения приобрели требуемый облик, пришло время добавить новые ресурсы, определяющие окно свойств приложения. Чтобы создать ресурсы диалогового окна для каждой из вкладок свойств, выполните следующие действия.

1. Для создания нового ресурса диалогового окна щелкните на пиктограмме New Dialog панели инструментов Resource или нажмите клавиши <Ctrl+1>. В окне редактора диалоговых окон будет создано новое окно, которому будет присвоен идентификатор IDD_DIALOG1.

   Когда определение параметров этого диалогового окна будет завершено, оно станет первой вкладкой в окне свойств.

2. Удалите из окна кнопки OK и Cancel, для чего, щелкнув мышью, выберите каждую из них и нажмите клавишу <Del>.

3. Если на экране окно Properties еще не раскрыто, раскройте его, выбрав команду View⇨Properties. Присвойте идентификатору окна новое значение IDD_PAGE1DLG и измените текст заголовка на Page 1, как это показано на рис. 12.7.

4. В окне свойств диалогового окна щелкните на корешке вкладки Styles. В списке Styles выберите значение Child, а в списке Border — значение Thin. Сбросьте флажок опции System Menu. Диалоговое окно определения свойств примет вид, показанный на рис. 12.8.

   Стиль Child необходимо использовать по той причине, что вкладка свойств является для окна свойств дочерним окном.

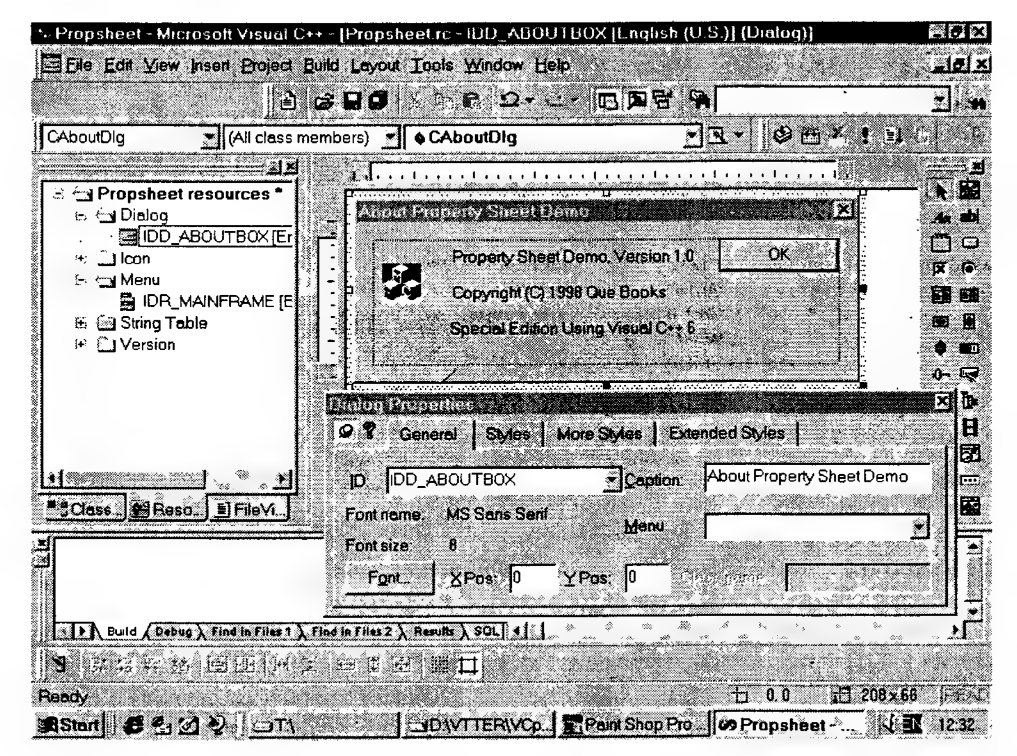


Рис. 12.5. Окончательный вид окна About

5. Поместите на вкладку текстовое поле ввода, как показано на рис. 12.9. В большинстве приложений вам потребуется изменить идентификатор этого ресурса — IDC_EDIT1 — на какой-либо иной, но в нашем демонстрационном приложении оставьте его без изменений.

6. Создайте вторую вкладку окна свойств, повторив пп. 1–5. Идентификатор этой вкладки определите как IDD_PAGE2DLG, а в качестве заголовка введите Page 2. Вместо текстового поля поместите на эту вкладку флажок опции, как показано на рис. 12.10.

Связывание ресурсов с классами

Итак, все необходимые ресурсы созданы. Далее требуется связать ресурсы двух новых вкладок окна свойств с классами C++, что даст возможность разрабатываемой программе управлять этими вкладками. Отдельный класс необходим и для самого окна свойств, которое будет содержать обе эти вкладки. Для определения новых классов выполните следующие операции.

1. Убедитесь, что в области редактирования диалогового окна отображена вкладка окна свойств Page 1, а затем сделайте на ней двойной щелчок. Возможный вариант: в меню выберите команду ClassWizard⇨View. Раскроется окно свойств MFC ClassWizard с установленной вкладкой Adding a Class, уже обсуждавшееся в главе 2.

2. Установите опцию Create New Class и щелкните на кнопке OK. Раскроется диалоговое окно New Class.

Рис. 12.6. Первый сегмент строки `IDR_MAINFRAME` *выводится на панель заголовка главного окна приложения*

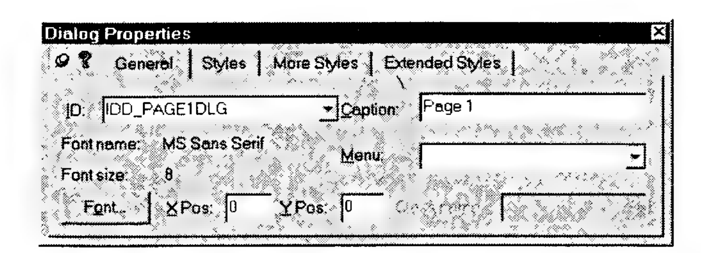


Рис. 12.7. Измените заголовок и идентификатор ресурса нового диалогового окна в окне Dialog Properties

3. В поле ввода Name введите значение `CPage1`. а в списке Base Class выберите значение `CPropertyPage`. (Будьте внимательны. не выберите случайно значение `CPropertySheet`.) Затем для создания указанного класса щелкните на кнопке OK.

   Только что вкладка окна свойств была связана с объектом класса `CPropertyPage`, что дает нам возможность манипулировать вкладкой, пользуясь этим объектом. Класс `CPropertyPage` приобретет для нас особую важность при изучении процесса создания мастеров.

4. В окне свойств MFC ClassWizard выберите вкладку Member Variables. Выделите значение `IDC_EDIT1` и щелкните на кнопке Add Variable. Раскроется диалоговое окно Add Member Variable.

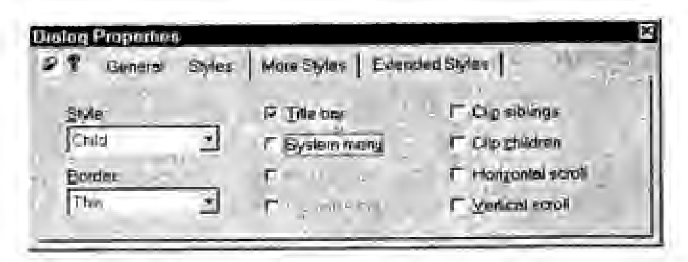


Рис. 12.8. Для вкладки окна свойств используются стили, отличные от тех, которые задаются для обычных диалоговых окон

Рис. 12.9. Вкладка окна свойств может содержать любые элементы управления

5. Присвойте переменной имя m_edit, как показано на рис. 12.11, а затем щелкните на кнопке OK. ClassWizard определит новую переменную-член, которая во вновь созданном классе будет содержать значение параметра элемента управления, помещенного на вкладку окна свойств.
6. Щелкните на кнопке OK, и определение класса CPage1 в окне свойств MFC ClassWizard Properties будет завершено.
7. Повторите пп. 1–7 для второй вкладки окна свойств. Присвойте новому классу имя CPage2 и создайте в нем логическую переменную-член с именем m_checkbox для элемента управления IDC_CHECK1, как показано на рис. 12.12.

Рис. 12.10. Такой вид должна иметь вторая вкладка окна свойств создаваемого приложения

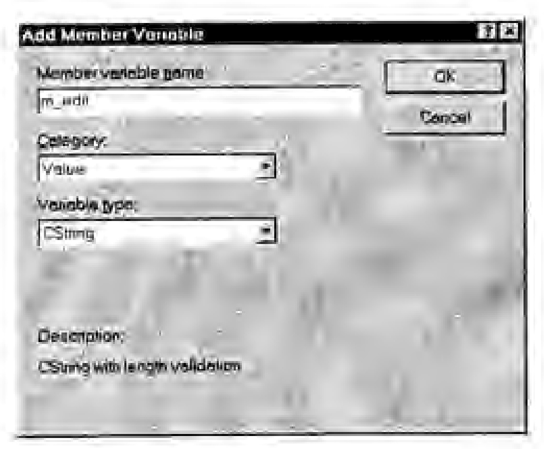


Рис. 12.11. С помощью ClassWizard можно подключить элементы управления диалогового окна к переменным-членам класса этого окна

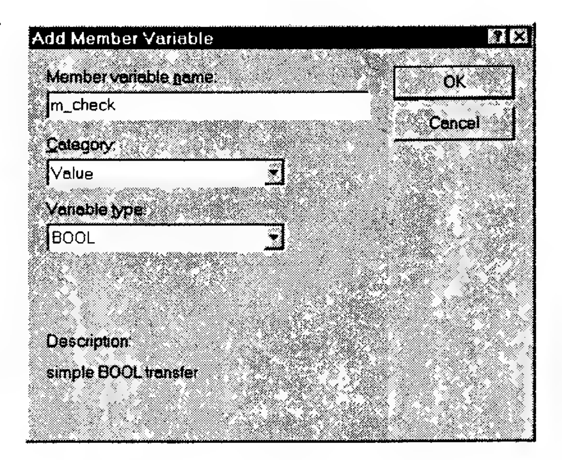


Рис. 12.12. Для второй вкладки окна свойств необходима логическая переменная-член m_checkbox, которая создается в диалоговом окне Add Member Variable

Создание класса для окна свойств

На данный момент вся работа по редактированию ресурсов уже выполнена, и нам больше не требуется на экране так много открытых окон. Выберите на панели меню команду Window⇨Close All и закройте окно свойств. Теперь необходимо создать класс для окна свойств, в котором будут отображаться определенные выше вкладки. Выполните следующие действия.

1. Выведите на экран окно ClassWizard и щелкните на кнопке Add Class. Под кнопкой будет раскрыто крошечное меню; выберите в нем команду New. На экране раскроется диалоговое окно New Class.
2. В поле ввода Name введите значение CPropSheet, а в списке Base Class выделите значение CPropertySheet. Щелкните на кнопке OK.
3. ClassWizard создаст класс CPropSheet. Для завершения определения класса в окне MFC ClassWizard Properties щелкните на кнопке OK.

Теперь у нас в программе определены три новых класса — CPage1, CPage2 и CPropSheet. Первые два класса являются производными от класса MFC CPropertyPage, а третий класс является производным от класса CPropertySheet. Хотя ClassWizard уже создал для этих классов файлы с заготовкой текста программ, требуется внести в них некоторые дополнительные элементы, обеспечивающие такое поведение классов, которое нам необходимо. Для завершения приложения Property Sheet Demo выполните следующие операции.

1. Раскройте окно просмотра классов, для чего щелкните на корешке вкладки ClassView. Разверните список классов группы Propsheet, как показано на рис. 12.13.
2. Сделайте двойной щелчок на имени класса CPropSheet для открытия файла заголовка, подготовленного для класса окна свойств. Поскольку имя этого класса (CPropSheet) очень схоже с именем всего приложения в целом (PropSheet), файлу заголовка присвоено имя PropSheet1 и в нем вы найдете определение класса CPropSheet, сгенерированного ClassWizard.

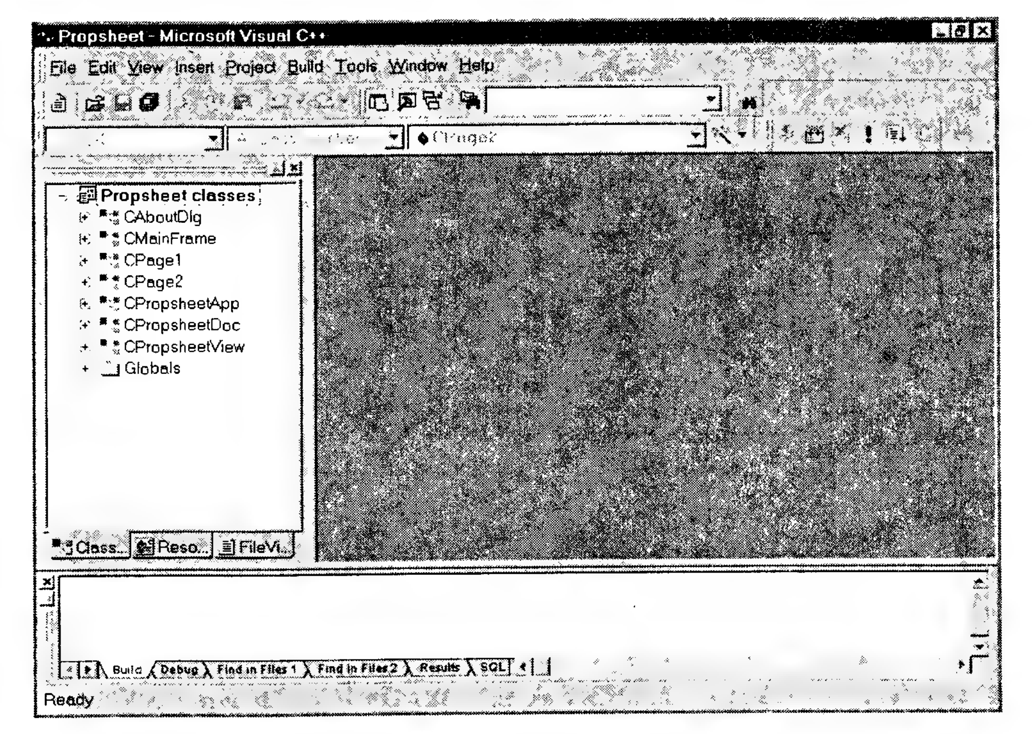


Рис. 12.13. В окне **ClassView** *перечислены все классы, определенные в приложении*

3. Добавьте следующие строки примерно в середине файла, непосредственно перед объявлением класса `CPropSheet`:

```
#include "page1.h"
#include "page2.h"
```

   Эти директивы обеспечат классу `CPropSheet` доступ к определению классов `CPage1` и `CPage2`, что даст возможность объявить в классе окна свойств переменные-члены этих классов.

4. Добавьте следующие операторы в секцию `// Attributes` класса `CPropSheet` сразу после ключевого слова `public`:

```
CPage1 m_page1;
CPage2 m_page2;
```

   Эти операторы объявляют члены-переменные класса, которые являются вкладками окна свойств и будут в нем отображаться.

5. Разверните класс `CPropSheet` на вкладке **ClassView** и сделайте двойной щелчок на элементе, обозначающем первый конструктор. Добавьте в него следующие строки:

```
AddPage(&m_page1);
AddPage(&m_page2);
```

   Это приведет к тому, что при построении в окно свойств будут включены две вкладки. Добавьте эти же строки во второй конструктор, расположенный сразу после первого.

6. В окне **ClassView** сделайте двойной щелчок на элементе `CPropsheetView` и отредактируйте соответствующий файл заголовка, добавив в его секцию `// Attributes` сразу после строки `CPropsheetDoc* GetDocument()` следующие строки:

```
protected:
   CString m_edit;
   BOOL m_check;
```

Эти строки объявляют две переменные-члены класса представления, предназначенные для хранения установок, выполненных пользователем в окне свойств.

7. В конструктор CPropertyView добавьте следующие строки:

```
m_edit = "Default";
m_check = FALSE;
```

Эти строки инициализируют значения в членах-данных экземпляра класса, так что при отображении окна свойств на экране принимаемые по умолчанию значения могут быть скопированы в элементы управления окна свойств. При изменении пользователем значений, установленных в окне свойств, в этих переменных-членах всегда будут содержаться последние установленные пользователем значения, что обеспечивает при необходимости возможность их восстановления.

8. Отредактируйте функцию CPropsheetView::OnDraw() так, чтобы она имела вид, показанный в листинге 12.1. Дополнительные строки предназначены для отображения на вкладках окна свойств текущих установок. После запуска программы будут отображаться значения, принятые по умолчанию.

Листинг 12.1. Функция CPropsheetView::OnDraw()

```
void CPropsheetView::OnDraw(CDC* pDC)
{
  CPropsheetDoc* pDoc = GetDocument();
  ASSERT_VALID(pDoc);

  pDC->TextOut(20, 20, m_edit);
  if (m_check)
    pDC->TextOut(20, 50, "TRUE");
  else
    pDC->TextOut(20, 50, "FALSE");
}
```

9. В начале файла реализации PropSheetView.cpp после всех директив #include вставьте

```
#include "propsheet1.h"
```

10. Перейдите в окно ClassWizard, щелкните на корешке вкладки Message Maps и убедитесь, что в поле Class Name выбрано значение CPropsheetView. Выберите в списке Object IDs значение IDD_PROPSHEET. Это значение является идентификатором новой команды, которая была помещена в меню File. Для добавления новой функции, которая будет обрабатывать командное сообщение, сгенерированное при выборе пользователем этой команды, щелкните на кнопке Add Function. Присвойте функции имя OnPropsheet(), как показано на рис. 12.14.

    Теперь функция OnPropsheet() связана с командой Property Sheet, помещенной нами ранее в меню File. Следовательно, когда пользователь выберет команду File⇨Property Sheet, MFC вызовет функцию OnPropsheet(), которая и должна будет выполнить все необходимые действия.

11. Для ввода текста функции OnPropsheet() щелкните на кнопке Edit Code. Поместите в текст функции строки, показанные в листинге 12.2.

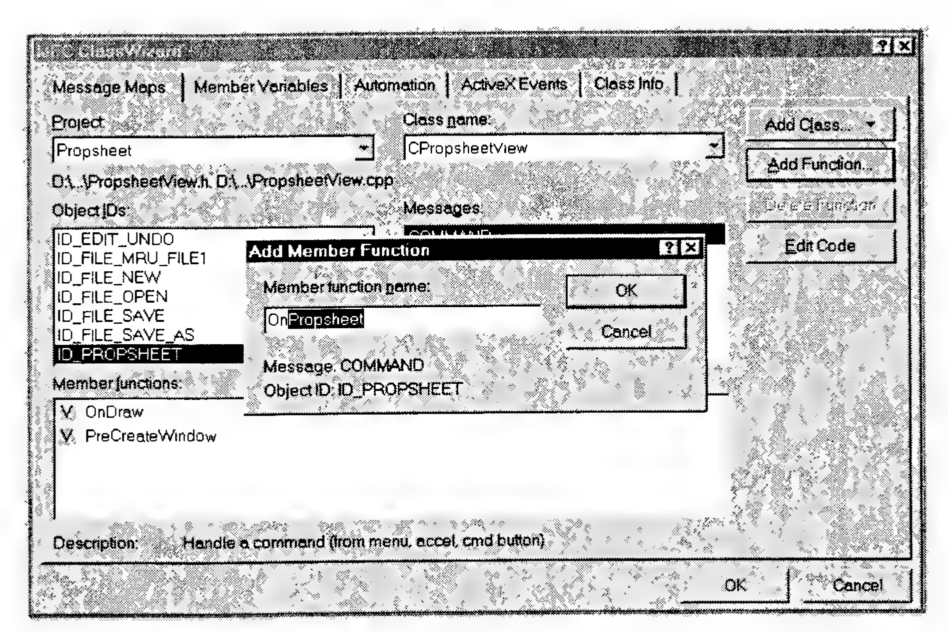


Рис. 12.14. В окно **MFC ClassWizard** *добавьте функцию-член* `OnPropsheet()`

Листинг 12.2. Функция `CPropsheetView::OnPropsheet()`

```
void CPropsheetView::OnPropsheet()
{
  CPropSheet propSheet("Property Sheet", this, 0);
  propSheet.m_page1.m_edit = m_edit;
  propSheet.m_page2.m_checkbox = m_check;
  int result = propSheet.DoModal();
  if (result == IDOK)
  {
    m_edit = propSheet.m_page1.m_edit;
    m_check = propSheet.m_page2.m_checkbox;
    Invalidate();
  }
}
```

В тексте фрагмента программы, показанного в листинге 12.2 (он будет подробно обсуждаться чуть ниже в этой же главе), создается экземпляр класса `CPropSheet` и присваиваются значения переменным-членам каждой из вкладок его окна. Вывод окна на экран производится обычной функцией `DoModal()`, обсуждавшейся нами в главе 2. Если пользователь щелкнет на кнопке OK, для отражения проведенных на каждой из вкладок изменений будет выполнено обновление переменных-членов представления, а затем функция `Invalidate()` организует перерисовку экрана.

Запуск приложения `Property Sheet Demo`

Разработка демонстрационного приложения полностью завершена. Откомпилируйте и скомпонуйте его, щелкнув на пиктограмме Build панели инструментов Build (или выбрав команду Build⇨Build). Запустите вновь созданное приложение, выбрав команду Build⇨Execute или щелкнув на пиктограмме Execute панели инструментов Build. На экране откро-

ется окно приложения, показанное на рис. 12.15. В этом окне представлены два значения, которые были приняты по умолчанию для элементов управления в окне свойств приложения. Можно изменить эти значения с помощью окна свойств. Выберите команду File⇨Property Sheet — на экране откроется окно свойств (рис. 12.16). Это окно свойств имеет две вкладки, каждая из которых содержит по одному элементу управления. Если вы измените назначения, установленные для этих элементов управления, и щелкнете на кнопке OK, в окне приложения будут выведены обновленные значения. Попробуйте!

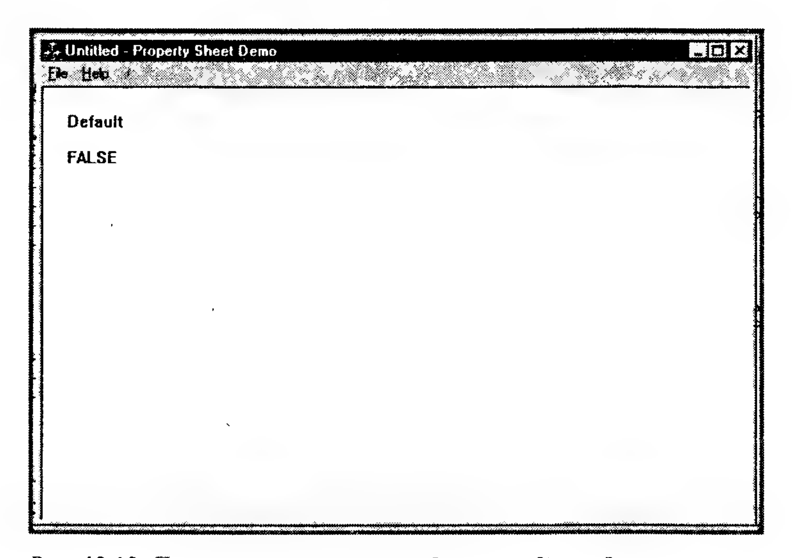


Рис. 12.15. При запуске приложения Property Sheet Demo *в его окне отображаются значения, принимаемые по умолчанию для элементов управления на вкладках окна свойств этого приложения*

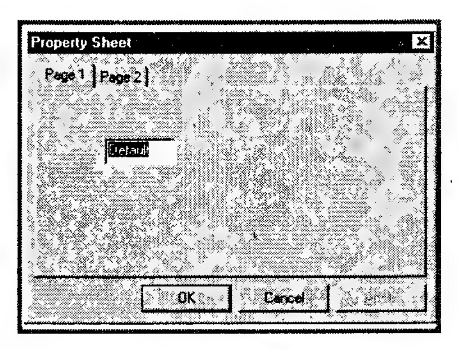


Рис. 12.16. Окно свойств приложения содержит две вкладки

Добавление окон свойств к приложениям

Чтобы добавить окно свойств к какому-либо из ваших собственных приложений, необходимо выполнить операции, сходные с теми, которые мы выполнили в предыдущем разделе при создании приложения-примера.

1. Создайте ресурсы диалогового окна для каждой из вкладок окна свойств. Для этих ресурсов должны быть определены стили Child и Thin, а системное меню должно отсутствовать.

2. Свяжите каждый из ресурсов вкладки окна свойств с объектом класса CPropertyPage. Эта задача легко решается с помощью ClassWizard. Подключите элементы управления на вкладке окна свойств к членам созданного класса.

3. Создайте с помощью ClassWizard класс для окна свойств. Он должен быть производным от класса MFC CPropertySheet.

4. Добавьте в класс окна свойств переменную-член для каждой помещаемой в это окно вкладки. Эти переменные-члены должны быть экземплярами классов вкладок окна свойств, созданных после выполнения п. 2.

5. Поместите в конструктор окна свойств вызов функции AddPage() для каждой из вкладок этого окна.

6. Для вывода на экран окна свойств следует вызвать его конструктор, а затем вызвать функцию-член класса окна свойств DoModal(), как это делается и в обычных диалоговых окнах.

После того как вы создали приложение и определили ресурсы и классы, представляющие окно свойств (или окна, если у вас их больше одного), следует определить способ, с помощью которого пользователь мог бы отобразить окно свойств, когда оно ему потребуется. В приложении Property Sheet Demo эта задача была решена посредством связывания команды меню с соответствующей функцией обработки сообщения. Но каким бы образом команда на отображение окна свойств ни была выдана, процесс вывода этого окна на экран всегда остается одним и тем же. Прежде всего следует вызвать конструктор класса окна свойств. В приложении Property Sheet Demo это делается так:

```
CPropSheet propSheet("Property Sheet", this, 0);
```

Этой командой в программе создастся экземпляр класса CPropSheet. Экземпляр (или объект) получает имя propSheet. Три аргумента, передаваемые конструктору, — строка заголовка, указатель на родительское окно (которое в данном случае является окном представления) и номер (считая от 0) вкладки, которая должна быть отображена в окне. Поскольку окна свойств создаются конструкторами этих окон, процедура создания окна свойств включает и создание его вкладок.

Создав объект окна свойств, можно инициализировать его члены-переменные, содержащие значения элементов управления вкладок окна свойств. В приложении Property Sheet Demo это делается таким образом:

```
propSheet.m_page1.m_edit = m_edit;
propSheet.m_page2.m_checkbox = m_check;
```

Теперь все готово для отображения окна свойств на экране. Вывод окна на экран выполняется точно так же, как и в случае обычных диалоговых окон — посредством вызова DoModal(), метода класса окна свойств:

```
int result = propSheet.DoModal();
```

Функция DoModal() не требует никаких аргументов, но возвращает значение, указывающее, на какой из кнопок щелкнул пользователь для выхода из окна свойств. В случае окон свойств или диалоговых окон обработка информации, введенной пользователем в элементы управления окна, потребуется, вероятнее всего, только если пользователь завершил работу в окне, щелкнув на кнопке OK. Такое завершение работы в окне индицируется возвращением значения IDOK. Если пользователь завершил работу в окне, щелкнув на кнопке Cancel, все выполненные им изменения игнорируются и переменные-члены класса представления или документа сохраняют прежние значения.

Преобразование окна свойств в мастер

Сейчас я сообщу вам нечто такое, что вызовет удивление у большинства: мастер, по сути, является просто специализированным окном свойств. В отличие от обычных окон свойств, вкладки которых допускают ввод информации пользователем в любом порядке и даже пропуск определенных вкладок целиком, мастера имеют кнопки Back (Назад), Next (Далее) и Finish (Готово), предназначенные для внесения определенного порядка в работу пользователя. Эта жесткая упорядоченность превращает мастер в превосходный инструмент для организации прохождения пользователем приложения последовательных этапов решения комплексной задачи. Из собственного опыта вы уже знаете, насколько мастер AppWizard в Visual C++ упрощает выполнение процедуры создания нового проекта. Можно разработать собственные мастера, предназначенные для работы в любых вновь создаваемых приложениях. Из последующих разделов этой главы вы узнаете, как обычное окно свойств очень просто преобразуется в мастер.

Приложение Wizard Demo

Приложение Wizard Demo поможет вам поближе познакомиться с технологией программирования мастеров. В целом это приложение аналогично приложению Property Sheet Demo, созданному нами в предыдущих разделах этой главы. В текст данной книги не включены пошаговые инструкции по созданию приложения Wizard Demo. Тем не менее при желании вы вполне сможете создать это приложение самостоятельно, основываясь на приведенном выше описании общих операций и воспользовавшись приводимыми ниже фрагментами текста программы.

Создание страниц мастера

Во всем, что касается ресурсов приложения, страницы мастеров создаются так же, как и вкладки окна свойств: методом создания диалогового окна с последующим переопределением его стилей. (Заголовки диалоговых окон — Page 1 of 3, Page 2 of 3 и Page 3 of 3 — указываются непосредственно в соответствующих окнах.) Каждый из ресурсов диалогового окна связывается с объектом класса CPropertyPage. Далее, чтобы получить контроль над страницами создаваемого мастера и иметь возможность отслеживать действия пользователя в его окне, необходимо переопределить функции OnSetActiv(), OnWizardBack(), OnWizardNext() и OnWizardFinish() в классах свойств страниц мастера. В дальнейшем мы рассмотрим, как это делается.

Вывод окна мастера на экран

Команда File⇨Wizard перехватывается функцией OnFileWizard() класса CWizView. Она очень похожа на функцию OnPropSheet()приложения Property Sheet, что видно из листинга 12.3. Первое отличие состоит в вызове функции SetWizardMode() перед вызовом функции DoModal(). Вызов этой функции сообщает MFC о том, что он должен отобразить данное окно свойств как окно мастера, а не как обычное окно свойств. Второе, и последнее, отличие состоит в том, что изменения, введенные пользователем на страницах мастера, будут приняты, если пользователь сделает щелчок на кнопке Finish, а не на кнопке OK. По этой причине значение, возвращаемое из функции DoModal(), проверяется на равенство с ID_WIZFINISH вместо IDOK.

Листинг 12.3. Функция CWizView::OnFileWizard()

```
void CWizView::OnFileWizard()
{
  CWizSheet wizSheet("Sample Wizard", this, 0);
  wizSheet.m_page1.m_edit = m_edit;
  wizSheet.m_page2.m_check = m_check;
  wizSheet.SetWizardMode();
  int result = wizSheet.DoModal();
  if (result == ID_WIZFINISH)
  {
    m_edit = wizSheet.m_page1.m_edit;
    m_check = wizSheet.m_page2.m_check;
    Invalidate();
  }
}
```

Определение кнопок мастера

Сразу же после отображения на экране очередной страницы мастера MFC автоматически вызывает метод OnSetActive(). Поэтому, когда программа отображает на экране первую страницу мастера, вызывается функция OnSetActive() класса CPage1. В эту функцию следует добавить программный код, который сделает поведение мастера именно таким, какое нам требуется. Текст функции CPage1::OnSetActive() показан в листинге 12.4.

Листинг 12.4. Функция CPage1::OnSetActive()

```
BOOL CPage1::OnSetActive()
{
  CPropertySheet* parent = (CPropertySheet*)GetParent();
  parent->SetWizardButtons(PSWIZB_NEXT);
  return CPropertyPage::OnSetActive()
}
```

Прежде всего функция OnSetActive() получает указатель на окно свойств мастера, которое является родительским окном для окон страниц. Затем она вызывает функцию мастера SetWizardButtons(), которая определяет состояние кнопок на странице. Функция SetWizardButtons() принимает единственный аргумент, который является набором значений флажков, указывающих, как кнопки должны отображаться на странице. Этими флажками являются: PSWIZB_BACK, PSWIZB_NEXT, PCWIZB_FINISH и PSWIZB_DISABLEDFINISH. Поскольку в листинге 12.4 вызов функции SetWizardButton() включает только флажок PSWIZB_NEXT, на данной странице будет активна только кнопка Next.

Поскольку вторую страницу мастера представляет класс CPage2, определенный в нем вызов функции SetWizardButtons() задействует как кнопку Back, так и кнопку Next, комбинируя соответствующие флажки с помощью терминального оператора побитового *ИЛИ* (|):

```
parent->SetWizardButtons(PSWIZB_BACK | PSWIZB_NEXT);
```

Поскольку третья страница мастера является последней, в классе CPage3 функция SetWizardButtons() вызывается следующим образом:

```
parent->SetWizardButtons(PSWIZB_BACK | PSWIZB_FINISH);
```

Данный набор флажков задействует кнопки Back и Finish вместо кнопки Next.

Обработка сообщений от кнопок мастера

В самом простом случае MFC позаботится обо всем, что должно быть сделано с целью перехода с данной страницы мастера на следующую (т.е. MFC сам предпринимает все необходимые действия для выполнения команд Back, Next, Finish и Cancel, когда пользователь делает щелчок на кнопке). Однако иногда вам придется самостоятельно выполнить некоторые нестандартные действия после щелчка пользователя на одной из кнопок. Например, провести контроль корректности установок, определенных пользователем на текущей странице мастера. Если будут обнаружены ошибки, можно потребовать от пользователя исправить их, прежде чем предоставить ему возможность перейти на другую страницу.

Чтобы самостоятельно обрабатывать щелчки пользователя на кнопках мастера, необходимо переопределить функции-члены OnWizardBack(), OnWizardNext() и OnWizardFinish(). Сделать это можно на вкладке Message Maps в окне ClassWizard. Имена указанных функций вы найдете в окне Messages в том случае, когда в списке Class name выбрано имя одного из классов страниц мастера. Когда пользователь сделает щелчок на кнопке мастера, MFC вызовет соответствующую функцию, в которой можно запрограммировать любые действия, необходимые для обработки данных на текущей странице мастера. Примером подобных действий в приложении Wizard Demo является запрещение пользователю перехода со второй страницы мастера на следующую, если флажок опции в этом окне не установлен. Действия эти осуществляются путем перегрузки функций OnWizardBack() и OnWizardNext(), как показано в листинге 12.5.

Листинг 12.5. Функции обработки сообщений от кнопки мастера

```
LRESULT CPage2::OnWizardBack()
{
  CButton *checkBox = (CButton*)GetDlgItem(IDC_CHECK1);
  if (!checkBox->GetCheck())
  {
    MessageBox("You must check the box.");
    return -1;
  }
  return CPropertyPage::OnWizardBack();
}

LRESULT CPage2::OnWizardNext()
{
  UpdateData();
  if (!m_check)
  {
    MessageBox("You must check the box.");
    return -1;
  }
  return CPropertyPage::OnWizardNext();
}
```

Эти функции демонстрируют два метода анализа состояния флажка опции на второй странице мастера. Функция OnWizardBack() получает указатель на окно флажка опции посредством вызова функции GetDlgItem(). Получив указатель, она вызывает метод GetCheck() класса элемента управления — флажка. Эта функция возвращает 1, если флажок установлен. Функция OnWizardNext() вызывает функцию UpdateData() для заполнения переменных-членов класса CPage2 значениями из элементов управления диалогового окна, а затем анализирует состояние переменной m_check. В обеих функциях, если флажок опции не установлен, выводится окно сообщения и функция возвращает значение -1. Возвращаемое значение, рав-

ное -1, указывает MFC на необходимость игнорировать щелчок на кнопке мастера и не выполнять переход на другую страницу. Как видите, обработка различных условий для принятия решения о возможности перехода на страницу вперед или назад не представляет особых трудностей.

Работа с мастером

При запуске приложения Wizard Demo раскрывается его главное окно, очень похожее на главное окно приложения Property Sheet Demo. Здесь меню File включает команду Wizard (мастер), при выборе которой на экран будет выведено окно мастера, показанное на рис. 12.17.

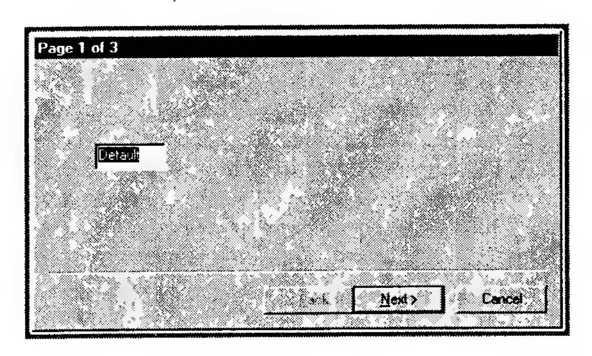


Рис. 12.17. Приложение Wizard Demo вместо окна свойств выводит на экран окно мастера

Окно этого мастера не блистает оформлением, но оно может продемонстрировать все, что вам следует знать для разработки более сложных мастеров. Как видно на рисунке, этот мастер имеет три страницы. На первой странице присутствуют текстовое поле и три кнопки — Back, Next и Cancel. Кнопка Back заблокирована, поскольку не существует предыдущих страниц, к которым можно было бы перейти. Кнопка Cancel предоставляет пользователю возможность в любой момент отказаться от продолжения работы с мастером и отменить всю обработку, уже выполненную в процессе работы с ним. Щелчок на кнопке Next вызовет отображение мастером его следующей страницы.

При желании можно ввести в поле ввода произвольный текст. После щелчка на кнопке Next события примут более интересный оборот. На экран будет выведена вторая страница мастера, показанная на рис. 12.18. Эта страница содержит поле некоторой опции и три стандартные кнопки — Back, Next и Cancel. В данном случае кнопка Back уже активизирована, поскольку появилась возможность при необходимости вернуться на страницу 1. Продолжим работу. Щелкните на кнопке Back. Мастер сообщит вам, что флажок опции должен быть установлен (рис. 12.19). Как вы увидите позднее, наличие у мастеров данной функции предоставляет вам возможность проверить содержание конкретной страницы, прежде чем разрешить пользователю сделать следующий шаг.

Установив флажок опции, можно щелкнуть на кнопке Back и вернуться на страницу 1 или щелкнуть на кнопке Next и перейти на страницу 3. После перехода на страницу 3 вы увидите на экране окно, показанное на рис. 12.20. Здесь кнопка Next заменена кнопкой Finish, поскольку вы находитесь на последней странице мастера. Если вы щелкнете на этой кнопке, работа с мастером будет завершена.

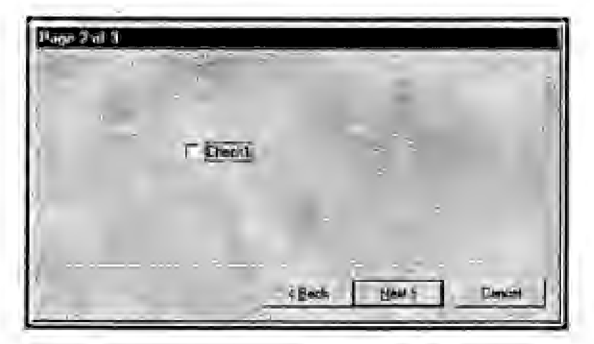


Рис. 12.18. На второй странице мастера кнопка Back *является активной*

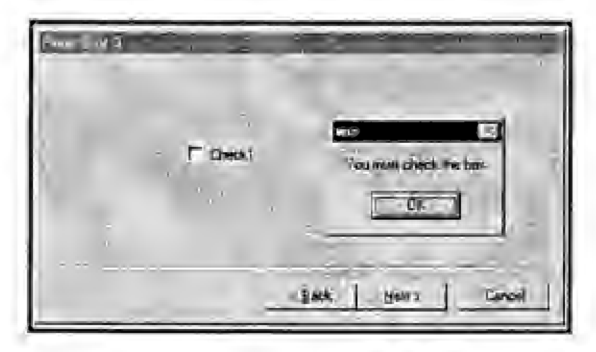


Рис. 12.19. Вы должны установить флажок опции, прежде чем мастер разрешит вам покинуть страницу 2

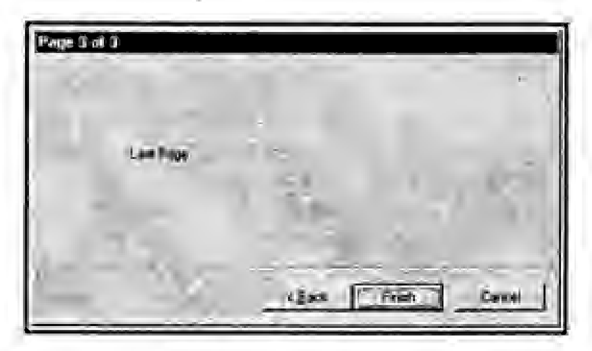


Рис. 12.20. Это последняя страница мастера, входящего в приложение `Wizard Demo`

ЧАСТЬ IV

Приложения и элементы управления ActiveX

В этой части...

ГЛАВА 13

Концепции технологии ActiveX

В этой главе...

Назначение технологии ActiveX

Связывание объектов

Внедрение объектов

Контейнеры и серверы

Совершенствование пользовательского интерфейса

Модель многокомпонентных объектов

Автоматизация в технологии ActiveX

Элементы управления ActiveX

Назначение технологии ActiveX

В этой главе рассматривается теория и концепция технологии ActiveX, которая построена на основе *модели COM* (Component Object Model — модель многокомпонентных объектов). До недавнего времени технология, построенная на основе COM, носила название OLE (Object Linking and Embedding — связывание и внедрение объектов), но теперь она обозначается термином ActiveX. Большинство программистов нового поколения используют OLE довольно осторожно, и замена названия на ActiveX, к несчастью, только укрепила осторожное к ней отношение. Тем не менее, если вы расцените технологию ActiveX как способ многократного использования программ, уже написанных и протестированных кем-то другим, и как защиту от необходимости самому повторно изобретать колесо в каждом новом приложении, вы поймете, что эта овчинка стоит выделки и времени на ее изучение. Наличие `Developer Studio` и MFC делает использование технологии ActiveX намного проще, поскольку они выполняют большую часть черновой работы, оставляя ее невидимой для вас. Часть IV состоит из пяти глав, в которых дается представление о современном состоянии технологии ActiveX. Кроме того, некоторые вопросы использования технологии ActiveX рассматриваются в главах 20 и 21. К изучению их лучше приступать после усвоения материала глав 18 и 19.

Windows всегда обеспечивала пользователям возможность одновременного запуска нескольких приложений. И с самого начала существования этой системы программисты хотели иметь на вооружении методы, с помощью которых запущенные приложения могли бы обмениваться информацией в процессе выполнения. Прекрасной новацией стал буфер обмена Clipboard, но пользователь все еще должен был выполнять вручную многие операции. Механизм DDE (Dynamic Data Exchange — динамический обмен данными) предоставил приложениям возможность обмена данными, но при этом сохранял некоторые серьезные ограничения. Затем появилась технология OLE 1. Позднее ее сменила OLE 2, с течением времени переименованная Microsoft просто в OLE, и, наконец, теперь она получила название ActiveX.

На заметку

Опытные пользователи Windows, вероятно, хорошо знакомы с примерами, которые приводятся в начале данной главы. Если вы знаете, что именно технология ActiveX может предложить пользователю, и интересуетесь только тем, как все это функционирует, переходите сразу к разделу *Модель многокомпонентных объектов*, в котором завеса таинственности будет приподнята.

Технология ActiveX дает возможность пользователю и приложениям ориентироваться на работу с документами, и это, пожалуй, самое главное. Если пользователь решает подготовить годовой отчет с помощью приложений, поддерживающих ActiveX, он может сосредоточиться именно на отчете как таковом. Вероятно, часть этой работы будет выполнена в Word, а еще некоторая ее часть — в Excel, но для пользователя суть дела заключена не в приложениях. Подобная переориентация сейчас происходит во многих направлениях и вызвано это объектно-ориентированным образом мышления большинства программистов. Ныне кажется более естественным разделить работу между несколькими различными приложениями, способными взаимодействовать между собой, чем написать одно гигантское приложение на все случаи жизни.

Вашему вниманию предлагается простой тест, способный выявить присущий вам стиль мышления — ориентацию на документы или на приложения. Как организована информация, хранящаяся на жестком диске вашего компьютера?

Структура каталогов, представленная на рис. 13.1, ориентирована на приложения: папки носят названия приложений, используемых для хранения документов. Все документы Word собраны вместе.

Структура каталогов, представленная на рис. 13.2, является документо-ориентированной: папки носят имена проектов или клиентов, документы которых хранятся в них. Все файлы о

продажах собраны вместе, даже если доступ к ним осуществляется с помощью различных приложений.

Если вы достаточно долго имеете дело с персональными компьютерами, вы еще помните времена, когда приходилось работать с отдельными дискетами для программ и для данных. Вероятно, вы сможете припомнить и процедуры инсталляции программного обеспечения, в которых требовалось указать каталог, предназначенный для хранения всех файлов, созданных данным продуктом. Все это были яркие примеры подхода, ориентированного на приложения, в скором времени уступившего свои позиции документо-ориентированному подходу.

```
Microsoft Office
   Word
      Создание приложений для Internet
      Использование Visual C++
      Acme Corp
         Обучение
         Web-страницы
   Excel
      Журнал
      Продажи
      Входящие
   ABC Inc
      Платежи
      Склад
Microsoft Visual Studio
   Платежи для ABC Inc
   Склад для ABC Inc
```

Рис. 13.1. Структура каталогов, отражающая ориентацию на приложения, в которой документы упорядочиваются по типу

Но почему? Что является ошибочным в подходе, ориентированном на приложения? А вот что: куда вы поместите документ, одинаково часто обрабатываемый двумя разными приложениями? Были времена, когда каждый программный продукт умел считывать файлы только в своем собственном формате. Но сейчас границы между приложениями размыты. Документы, созданные в одном текстовом редакторе, легко могут быть прочитаны в другом; файл с электронной таблицей может быть обработан в базе данных и т.д. Если заказчик пошлет вам документ, созданный с помощью WordPerfect, а у вас нет этого приложения, станете ли вы для хранения данного файла создавать папку с именем `\WORDPERFECT\DOCS`, поместите его в папку `\MSOFFICE\WORD\DOCS` или поступите как-то иначе? Если папки на вашем жестком диске организованы в документо-ориентированной манере, вы сможете просто поместить данный файл в папку для этого клиента.

Интерфейс Windows 95, в настоящее время встроенный и в Windows NT, ориентирован на документо-ориентированный образ мышления, предоставляя пользователям возможность посредством двойного щелчка на документе автоматически запускать создавшее его приложение. В целом, это не ново — `File Manager` имеет такие возможности уже многие годы. Однако существует большая разница между двойным щелчком, выполненным на пиктограмме документа на рабочем столе и двойным щелчком на элементе в окне списка, хотя в обоих

случаях следствием будет запуск соответствующего приложения. Чем дальше, тем меньшее значение имеет, какое именно приложение (или приложения) применялось при создании документа — вам просто необходимо его просмотреть и как можно быстрее внести изменения.

```
Клиенты
  Acme Corp
    Обучение
    Web-страницы
    Входящие
  ABC Inc
    Платежи
    Склад
    Входящие
  Книги
    Создание приложений для Internet
    Использование Visual C++
    ...
  Прочее
    Счета
    Продажи
```

Рис. 13.2. Документо-ориентированная структура каталогов, в которой документы упорядочиваются по назначению или содержанию

Когда вы измените свой подход на документо-ориентированный, вы почувствуете притягательную силу составных документов — файлов, созданных несколькими приложениями. Если ваш отчет нуждается в иллюстрациях, вы создаете их с помощью какой-либо графической программы, а затем, когда они будут готовы, помещаете их в текст. Если в годовой отчет необходимо включить таблицу, данные для которой уже введены в электронную таблицу, не следует заново вводить и упорядочивать их, используя средства табличной обработки текстового редактора, или даже импортировать их. Поместите их в текст отчета непосредственно как фрагмент электронной таблицы. Все это, конечно, уже не ново. Даже ранние издательские системы для персональных компьютеров, такие как Ventura, умели объединять в один сложный составной документ текст и графику, получаемую из различных источников. Новизна заключатся в том, что теперь все это выполняется просто и быстро для различных приложений.

Связывание объектов

На рис. 13.3 показан документ Word, включающий связанную с ним электронную таблицу Excel.

Чтобы создать подобный документ собственными силами, выполните следующие действия.

1. Запустите Word и введите необходимый текст.
2. Щелкните в том месте документа, где должна появиться таблица.

3. Выберите команду Insert⇨Object (Вставка⇨Объект).
4. В раскрывшемся диалоговом окне выберите вкладку Create From File (Создание из файла).
5. Выберите или введите имя файла, как это делается в диалоговом окне File Open (Открытие документа).
6. Убедитесь, что флажок опции Link to File (Связь с файлом) установлен.
7. Щелкните на кнопке OK.

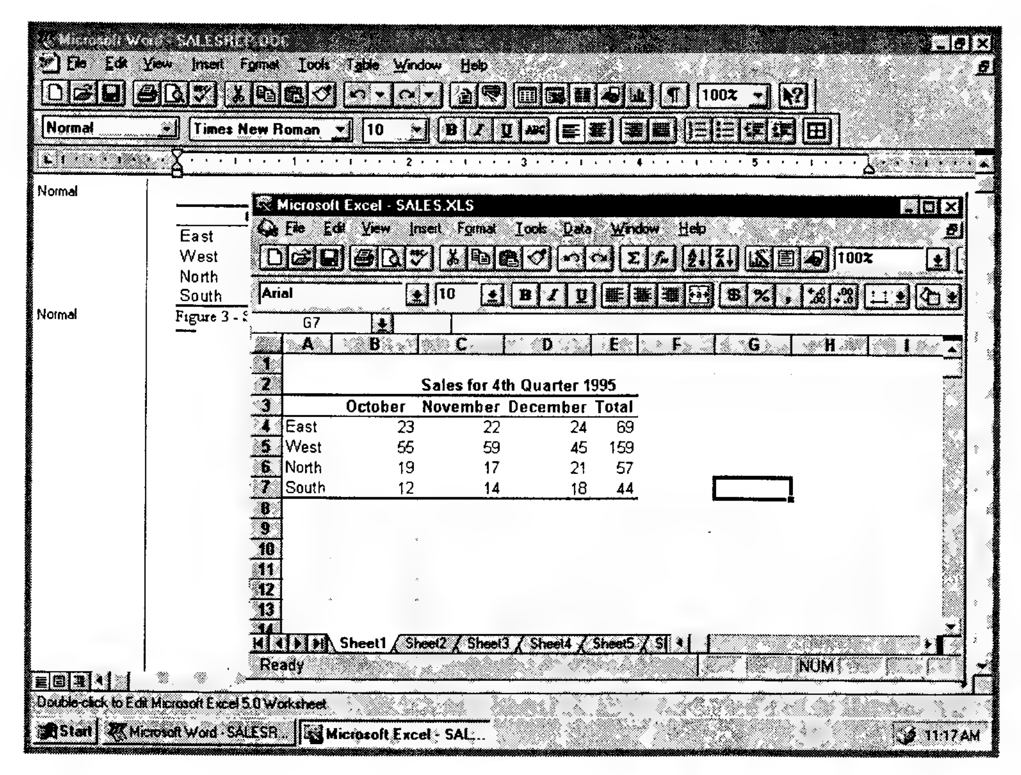


Рис. 13.3. Документ Microsoft Word может быть связан с файлом Excel

В вашем документе будет отображен весь файл электронной таблицы. Если в файле Excel на диске вы сделаете изменения, они будут отражены и в документе. Можно отредактировать электронную таблицу в ее собственном приложении, сделав двойной щелчок на ее изображении в документе Word. При этом для редактирования таблицы будет открыто отдельное окно Excel, как показано на рис. 13.4. Если исходный файл таблицы Excel удалить с диска, документ Word по-прежнему будет содержать ее изображение, но возможность редактирования этой таблицы будет утрачена.

Связывание в документах целесообразно в том случае, если планируется использовать связываемый файл во многих документах. При этом изменения, вносимые в файл, будут автоматически отражаться во всех документах, с которыми он был связан. Связывание не вызывает чрезмерного увеличения размеров документа, поскольку в последнем сохраняются лишь сведения о расположении связанного файла и немного дополнительной информации.

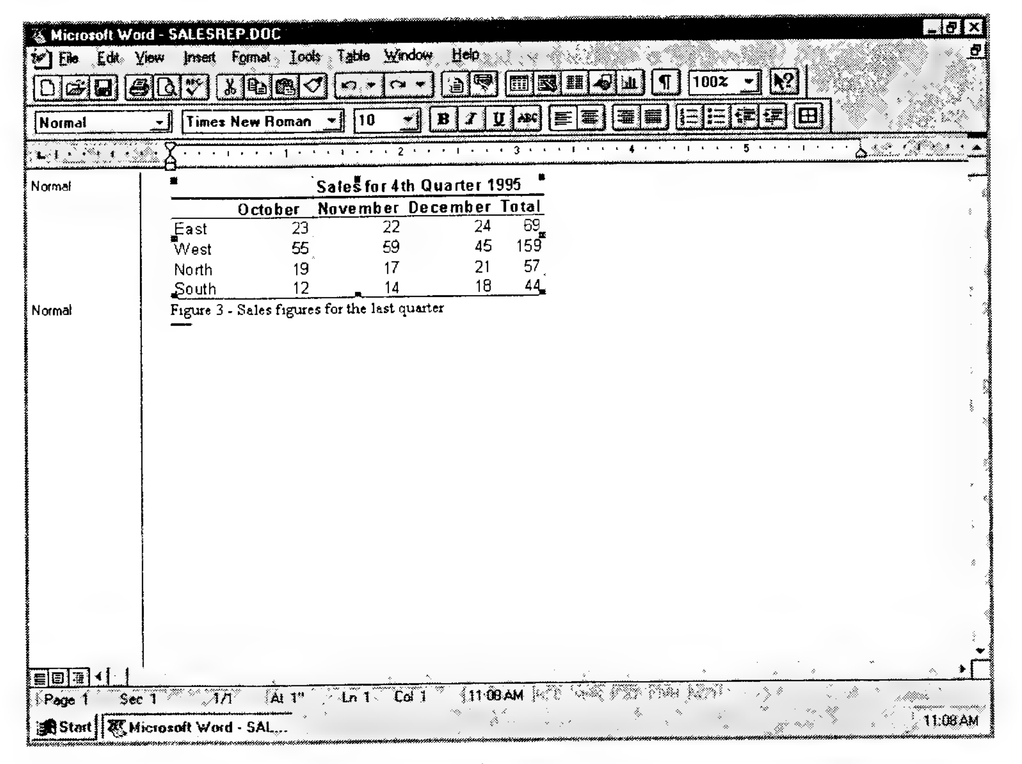


Рис. 13.4. Двойной щелчок на связанном объекте вызывает запуск приложения, в котором этот объект был создан

Внедрение объектов

Внедрение подобно связыванию, но при внедрении создается копия объекта, которая и помещается в составной документ. Если в дальнейшем откорректировать исходный файл, изменения не будут отражены в составном документе. По внешнему виду диаграммы Excel в составном документе Word нельзя определить, связан этот объект с документом или внедрен в него. На рис. 13.5 показана электронная таблица Excel, внедренная в документ Word.

Самостоятельно создать подобный составной документ можно, выполнив следующие действия.

1. Запустите Word и введите требуемый текст.
2. Щелкните мышью там, где требуется разместить таблицу.
3. Выберите команду Insert⇨Object.
4. В раскрывшемся диалоговом окне выберите вкладку Create From File.
5. Выберите или введите имя требуемого файла так же, как в диалоговом окне File Open.
6. Сбросьте флажок опции Link to File.
7. Щелкните на кнопке OK.

Вы видите какие-либо отличия? Их можно будет увидеть, дважды щелкнув на внедренном объекте с целью его редактирования. Меню и панели инструментов Word исчезнут, а их место будет занято соответствующими элементами Excel, как показано на рис. 13.6. Изменения,

которые вы сделаете в этом документе, не затронут исходного файла, данные из которого были внедрены в документ. Эти изменения коснутся только копии исходного файла, которая стала частью документа Word.

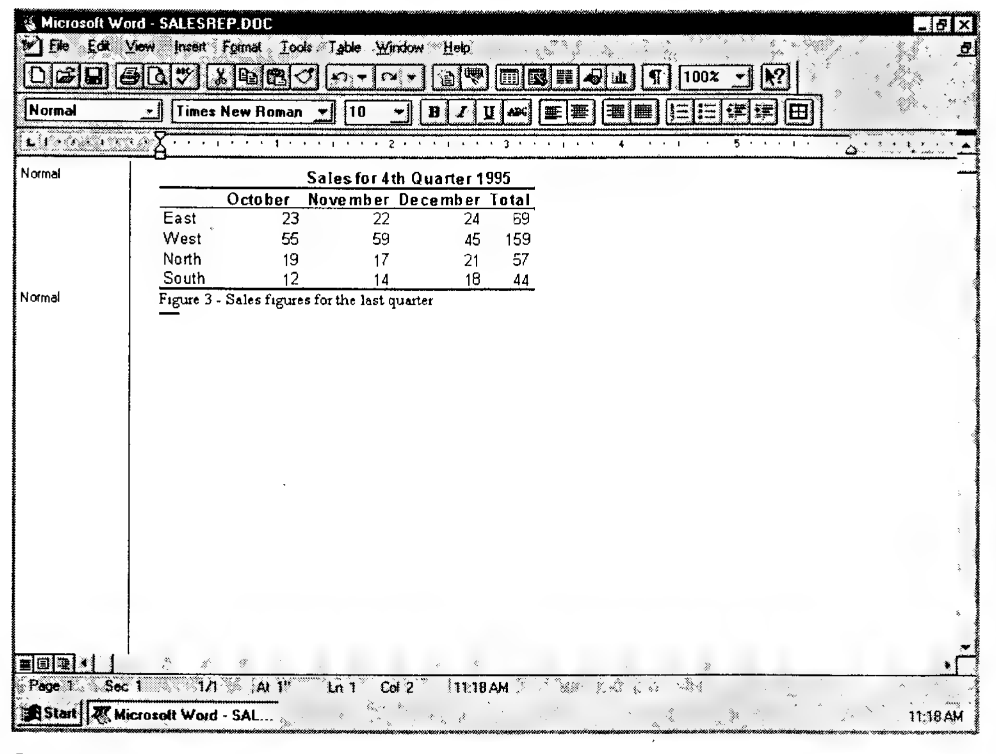


Рис. 13.5. Файл, внедренный в другой файл, выглядит точно так, как и файл, связанный с другим файлом

Внедрение объектов в документ следует проводить в том случае, если вы планируете создать составной документ и в дальнейшем работать над ним, как над единым целым, не обращаясь более к файлам отдельных составляющих его частей. Любые выполненные в документе изменения не повлияют на остальные файлы на диске, в том числе и на те, копии которых были помещены в составной документ. Внедрение значительно увеличивает размеры файлов документов, но вы всегда можете удалить оригиналы внедренных файлов, если объем свободного пространства на диске имеет для вас существенное значение.

Контейнеры и серверы

Чтобы выполнить внедрение или связывание двух объектов, вам необходимо иметь *контейнер* и *сервер*. Контейнером является приложение, в документ которого объект внедряется или с документом которого объект связывается (в приведенных выше примерах это Word). Сервером является приложение, в котором объект был создан и которое может быть запущено, когда на объекте будет сделан двойной щелчок (в нашем случае это Excel).

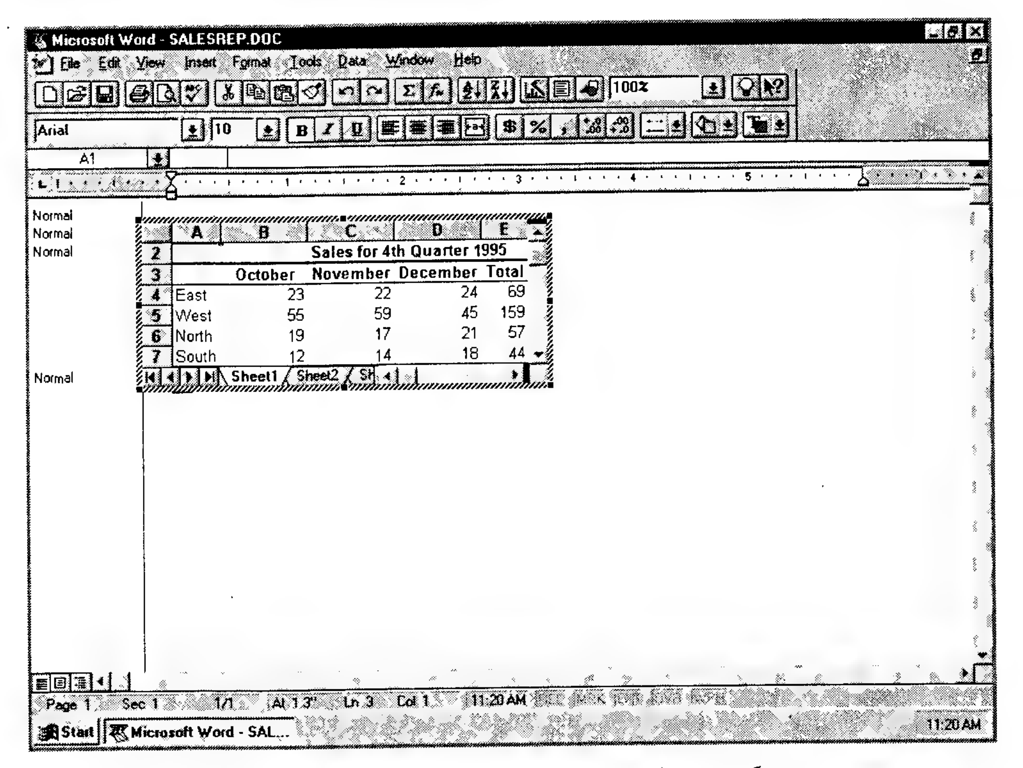


Рис. 13.6. Редактирование на месте используется при внедрении объектов по технологии OLE

Почему необходимо разрабатывать приложения как контейнеры ActiveX? Для сокращения объема работы, которую вам же придется выполнять. Представим, что вы уже завершили разработку приложения и передали его в эксплуатацию пользователю. С помощью этого приложения решается какая-то специфическая задача, например организация работы отдела сбыта или подготовка расписания игр в спортивной лиге, или вычисление страховых ставок. Через некоторое время пользователь обращается к вам с просьбой добавить к приложению некоторые функции, свойственные электронным таблицам, чтобы у него появилась возможность выполнять определенные вычисления непосредственно в этом приложении. Сколько времени вам потребуется для внесения соответствующих изменений? И есть ли вообще у вас время на изучение вопроса о том, как в программах-процессорах электронных таблиц обрабатываются математические функции, задаваемые пользователем?

Если приложение является контейнером ActiveX, времени вообще не потребуется. Порекомендуйте пользователю внедрять или связывать его документы с файлами Excel. Если у пользователя нет собственной копии Excel, он может использовать любое другое приложение обработки электронных таблиц, поддерживающее функции сервера ActiveX.

Однако никто не ограничивает нас только использованием электронных таблиц. Что если у пользователя возникнет необходимость сделать небольшие заметки? Пусть он использует вставку документов Word. А как быть с объектами растровой графики и прочими иллюстрациями? Воспользуемся Microsoft Paint или другим более мощным графическим пакетом, если он имеется у заказчика и может работать в режиме сервера ActiveX. Нет необходимости самостоятельно разрабатывать и дополнять приложение всеми мыслимыми средствами, поскольку можно просто сделать свое приложение контейнером ActiveX, и заказчик получит

возможность внедрять в создаваемые им документы все, что ему понадобится, без вашего участия.

Хорошо, ну а зачем создавать приложения-серверы ActiveX? Снова обратимся к причинам, побуждающим вас к созданию приложений-контейнеров. Многие пользователи обращаются к разработчикам с просьбой добавить в интересующее их приложение ту или иную функцию. В ответ им сообщается, что данная возможность будет доступна им уже сейчас, если они приобретут соответствующее готовое приложение — электронную таблицу, текстовый или графический редактор или еще что-нибудь, в чем нуждается этот пользователь, и что может функционировать как сервер ActiveX. Если и ваше приложение будет сервером ActiveX, оно будет пользоваться спросом у тех, кто хотел бы объединить его возможности с возможностями приложений-контейнеров.

Сервер и контейнер дают пользователю возможность создавать именно такие документы, в которых он нуждается. Данная концепция является значительным шагом в направлении создания блочного программного обеспечения и утверждения документо-ориентированного подхода в работе. И если вы хотите, чтобы создаваемые вами приложения несли на себе логотип Windows 95, они должны быть или сервером, или контейнером, или и тем, и другим одновременно. Но технология ActiveX включает в себя, помимо связывания и внедрения, еще и многое другое.

Совершенствование пользовательского интерфейса

Что если объект, который требуется внедрить, является не файлом, а лишь частью документа, уже открытого вами в данный момент? Надо полагать, вы знаете о возможности использования буфера обмена Clipboard для пересылки объектов ActiveX. Например, для вставки части электронной таблицы Excel в документ Word выполните следующие действия.

1. Запустите Excel.
2. Запустите Word.
3. В окне Excel выделите ту часть информации, которую необходимо скопировать.
4. Выберите команду Edit⇨Copy, и выделенный блок будет помещен в Clipboard.
5. Переключитесь в окно Word и выберите команду Edit⇨Paste Special.
6. В раскрывшемся диалоговом окне установите переключатель Paste.
7. В окне списка выберите значение `Microsoft Excel Worksheet`.
8. Убедитесь, что флажок опции Display as Icon сброшен.
9. В результате диалоговое окно должно приобрести вид, показанный на рис. 13.7.
10. Щелкните на кнопке OK.

В результате копия части электронной таблицы будет внедрена в документ Word. Если вы установите переключатель в положение Paste Link, изменения в электронной таблице немедленно отобразятся в документе Word, а не только в случае их сохранения на диск. (Для обновления этих данных в Word вам, возможно, потребуется сделать на них щелчок.) Это будет справедливо и в том случае, если документу электронной таблицы даже не было присвоено собственное имя и он ни разу не сохранялся на диске. Попробуйте сами! Это лучше, чем записывать на диск никчемные файлы только для того, чтобы после внедрения в составной документ сразу же их удалить.

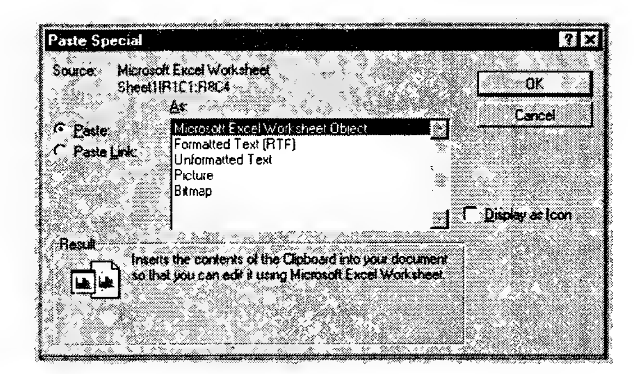


Рис. 13.7. Диалоговое окно Paste Special *используется для связывания или внедрения выделенной части документа*

Другой способ внедрнть часть одного документа в другой — использовать технологию *перетащить и опустить*. Эта технология работает в самых различных контекстах. Вы делаете щелчок на чем-либо (на пиктограмме, на выделенном блоке текста, выбранном элементе списка) и, удерживая кнопку мыши в нажатом состоянии, выполняете перемещение объекта. То, на чем был сделан щелчок, перемещается вслед за указателем мыши, а после того как вы отпустите ее кнопку, фиксируется в новой позиции. Это действие интуитивно понятно и логично, как изменение размера или перемещение диалоговых окон, но теперь область применения этой технологии стала намного шире. Например, вот как выглядит тот же самый обмен данными между Word и Excel, выполняемый с помощью технологии *перетащить и опустить*.

1. Запустите Word и установите размер его окна меньше размера полного экрана.
2. Запустите Excel и установите размер его окна также меньше размера полного экрана. Лучше всего, если вам удастся так расположить на экране окна Word и Excel, чтобы они не перекрывались.
3. В окне Excel выделите с помощью мыши или клавиш управления курсором фрагмент таблицы, предназначенный для копирования.
4. Щелкните на рамке выделенного фрагмента таблицы (толстой черной линии) и оставьте кнопку мыши нажатой.
5. Перетащите фрагмент в документ Word и отпустите кнопку мыши.

Выделенный блок будет внедрен в документ Word. После выполнения на нем двойного щелчка этот фрагмент можно будет отредактировать с помощью Excel. Перетаскивание можно применять и для перемещения или копирования выделенных фрагментов в пределах одного документа.



По умолчанию выделенный блок информации пересылается. Это означает, что из таблицы Excel он будет удален. Если вы хотите выполнить копирование, при перетаскивании удерживайте нажатой клавишу <Ctrl>, причем кнопку мыши следует отпустить раньше клавиши <Ctrl>.

Технологию *перетащить и опустить* можно применять и к пиктограммам. Если на рабочем столе перетащить файл в папку, будет выполнена его реальная пересылка в указанную папку. (Если при перетаскивании держать нажатой клавишу <Ctrl>, файл будет скопирован.) Если перетащить файл на пиктограмму программы, он будет в данной программе открыт. Это

очень полезное свойство, если у вас есть файлы, которые вы обрабатываете с помощью двух приложений. Например, страницы World Wide Web, являющиеся документами в формате HTML, часто создаются средствами редакторов HTML, но просматриваются с помощью броузеров World Wide Web, таких как Netscape Navigator. Если вы сделаете двойной щелчок на пиктограмме документа HTML, будет запущен броузер для его просмотра. Если вы перетащите пиктограмму документа на пиктограмму редактора HTML, данный документ будет открыт в окне указанного редактора. Реализация всех этих возможностей в повседневной работе за компьютером дает ощутимую экономию времени.

Программирование подобных функциональных возможностей в приложениях потребует от разработчика весьма незначительных дополнительных усилий. Так почему бы не воспользоваться этим?

Модель многокомпонентных объектов

Сердцем современной технологии ActiveX является COM — Модель многокомпонентных объектов (Component Object Model). Это очень сложная тема, которая вполне заслуживает отдельной книги. К счастью, Microsoft Foundation Classes и Visual C++ AppWizard берут на себя основную часть черновой работы, скрывая ее от ваших глаз. А потому обсуждение этой темы в данной главе даст вам вполне достаточный объем теоретических знаний, необходимых для разработчика приложений OLE.

Модель многокомпонентных объектов является двоичным стандартом интерфейса объектов в Windows. Это означает, что выполняемый программный код (в файлах .DLL или .EXE), который описывает объект, может быть вызван на выполнение другим объектом. Даже в том случае, когда оба объекта были написаны на разных языках, они сохраняют возможность взаимодействовать между собой, используя стандарт COM.

На заметку Поскольку программный код, хранимый в .DLL, выполняется в том же процессе, что и вызывающая его программа, это самый быстродействующий метод взаимодействия между приложениями. Когда два отдельных приложения взаимодействуют друг с другом по стандарту COM, вызовы функций одного приложения из другого должны быть упорядочены: COM собирает воедино все параметры и активизирует требуемую функцию своими средствами. По этой причине сервер, выполняемый как отдельное приложение (.EXE), работает медленнее, чем сервер, включенный в процесс (.DLL).

Но как они на самом деле взаимодействуют? Посредством *интерфейсов*. Интерфейс в ActiveX является набором функций или, точнее, имен функций. По сути, это класс C++, не имеющий данных и имеющий только виртуальные функции. Классы, создаваемые вами в программе, наследуют виртуальные функции от этого класса, но имеют собственный код для реализации каждой из них. (Вспомните, что класс, который наследует чисто виртуальную функцию, не наследует программного кода этой функции. Об этом более подробно сказано в приложении А.) Другие программы получат доступ к созданному вами программному коду посредством вызова этих функций. Все без исключения объекты ActiveX должны поддерживать интерфейс с именем `IUnknown` (и, как правило, поддерживают многие другие интерфейсы, имена которых всегда начинаются с буквы `I` — префикса имен интерфейсов).

Интерфейс `IUnknown` предназначен для единственной цели: получения доступа к другим интерфейсам. Он включает функцию, носящую имя `QueryInterface()`, которая получает на входе идентификатор (ID) интерфейса, а возвращает указатель на этот интерфейс в данном объекте. Все остальные интерфейсы наследуются от интерфейса `IUnknown` и поэтому также имеют функцию `QueryInterface()`, для которой вы должны разработать текст программы. И вам пришлось бы самостоятельно написать соответствующую подпрограмму, если бы не существовало библиотеки MFC. Последняя включает в себя множество макросов, которые су-

щественно упрощают создание интерфейсов и входящих в них функций (в этом вы очень скоро убедитесь на практике). Полное объявление интерфейса IUnknown приведено в листинге 13.1. Макросы, обеспечивающие определенную часть работы по объявлению интерфейсов, подробно здесь рассматриваться не будут. Объявление интерфейса IUnknown включает объявление трех функций: QueryInterface(), AddRef() и Release(). Последние две функции используются для отслеживания, какие приложения используют этот интерфейс в данный момент. Все три функции должны наследоваться всеми без исключения остальными интерфейсами, и реализация их должна быть представлена разработчиком интерфейса.

Листинг 13.1. Интерфейс IUnknown, определенный в файле \DevStudio\vc\include\unknwn.h

```
interface IUnknown
   {
   public:
      BEGIN_INTERFACE
      virtual HRESULT STDMETHODCALLTYPE QueryInterface(
         /* [in] */ REFIID riid,
         /* [iid_is][out] */ void __RPC_FAR *__RPC_FAR *ppvObject) = 0;

      virtual ULONG STDMETHODCALLTYPE AddRef( void) = 0;

      virtual ULONG STDMETHODCALLTYPE Release( void) = 0;

      END_INTERFACE
   };
```

Автоматизация в технологии ActiveX

Сервер автоматизации ActiveX обеспечивает другим приложениям возможность заказать ему, что требуется выполнить. Он *предоставляет* другим приложениям свои функции и данные, именуемые в этом случае *методами* и *свойствами*. Например, Microsoft Excel является сервером автоматизации ActiveX и программы, написанные на Visual C++ или Visual Basic, имеют возможность вызывать функции Excel и задавать значения свойств объектов, например ширину колонок. Все это означает, что для ваших приложений отпадает необходимость в создании языка сценариев. Если вы обеспечите возможность предоставления всех функций и свойств вашего приложения, любой язык программирования сможет стать языком сценариев для вашего приложения (если этот язык располагает средствами управления сервером автоматизации ActiveX). Пользователи и заказчики могут уже знать примененный вами язык сценариев. По сути, они будут избавлены от необходимости изучения тонкостей этого языка для создания собственных макросов и автоматизации своей работы с приложением (хотя им потребуется изучить имена методов и свойств, предоставляемых вашей программой).

Очень важным моментом, который следует учитывать при взаимодействии с объектами автоматизации ActiveX, является то, что в этом случае одна программа всегда является управляющей. Она вызывает методы или изменяет свойства другого выполняемого приложения. Управляющее приложение носит название *контроллера автоматизации ActiveX*. Приложение, предоставляющее свои методы и свойства, называется *сервером автоматизации ActiveX*. Excel, Word и прочие программы, входящие в состав Microsoft Office, выступают как серверы автоматизации ActiveX, и ваши программы могут непосредственно использовать функции этих приложений, что дает вам возможность реально сократить время написания программ.

Например, у вас есть возможность в создаваемых приложениях для обработки блока введенного пользователем текста использовать функцию программы Word, которая вызывается командой Format⇨Change Case (Формат⇨Регистр). В ваших руках будет инструмент, дающий возможность отформатировать текст, переведя все его буквы в верхний или нижний регистр, расставив регистр так, как в предложении (первая буква каждого предложения заглавная, остальные — прописные) или как в заголовке (первая буква каждого слова заглавная, остальные — прописные).

Описание того, как на самом деле работает автоматизация ActiveX, значительно сложнее краткого обзора интерфейса, сделанного в предыдущем разделе. Автоматизация предусматривает использование специального интерфейса, называемого IDispatch. Он является упрощенным интерфейсом, поддержка которого обеспечивается многими языками программирования, включая и Visual Basic, в котором не используются указатели. Объявление интерфейса IDispatch приводится в листинге 13.2.

Листинг 13.2. Интерфейс IDispatch, определенный в файле \Program Files\Microsoft Visual Studio\VC98\include\oaidl.h

```
MIDL_INTERFACE("00020400-0000-0000-C000-000000000046")
   IDispatch : public IUnknown
   {
   public:
      virtual HRESULT STDMETHODCALLTYPE GetTypeInfoCount(
         /* [out] */ UINT __RPC_FAR *pctinfo) = 0;

      virtual HRESULT STDMETHODCALLTYPE GetTypeInfo(
         /* [in] */ UINT iTInfo,
         /* [in] */ LCID lcid,
         /* [out] */ ITypeInfo __RPC_FAR *__RPC_FAR *ppTInfo) = 0;

      virtual HRESULT STDMETHODCALLTYPE GetIDsOfNames(
         /* [in] */ REFIID riid,
         /* [size_is][in] */ LPOLESTR __RPC_FAR *rgszNames,
         /* [in] */ UINT cNames,
         /* [in] */ LCID lcid,
         /* [size_is][out] */ DISPID __RPC_FAR *rgDispId) = 0;

      virtual /* [local] */ HRESULT STDMETHODCALLTYPE Invoke(
         /* [in] */ DISPID dispIdMember,
         /* [in] */ REFIID riid,
         /* [in] */ LCID lcid,
         /* [in] */ WORD wFlags,
         /* [out][in] */ DISPPARAMS __RPC_FAR *pDispParams,
         /* [out] */ VARIANT __RPC_FAR *pVarResult,
         /* [out] */ EXCEPINFO __RPC_FAR *pExcepInfo,
         /* [out] */ UINT __RPC_FAR *puArgErr) = 0;

   };
```

Хотя интерфейс IDispatch выглядит сложнее, чем IUnknown, он объявляет всего несколько дополнительных функций: GetTypeInfoCount(), GetTypeInfo(), GetIDsOfNames() и Invoke(). Поскольку он наследуется от IUnknown, он имеет и унаследованные функции QueryInterface(), AddRef() и Release(). Они являются чисто виртуальными, поэтому любой класс COM, наследующий от IDispatch, должен включать свою реализацию этих функций. Самой важной из всех определенных выше функций является Invoke(), используемая для вызова функций сервера автоматизации и доступа к его свойствам.

Элементы управления ActiveX

Элементы управления ActiveX являются миниатюрными серверами автоматизации ActiveX, которые загружаются и выполняются *в процессе*. Последнее указывает на то, что они работают исключительно быстро. Раньше их принято было называть элементами управления OLE. Они были разработаны для замены элементов управления VBX, 16-битовых элементов управления, написанных для использования в Visual Basic и Visual C++. (Имеется достаточное количество существенных технических причин, по которым технология VBX не может быть распространена на 32-битовые приложения.) Поскольку элементы управления OLE традиционно хранились в файлах с расширением `.OCX`, многие ссылались на элементы управления OLE, как на элементы управления OCX или просто OCX. Хотя технология OLE со временем была замещена ActiveX, создаваемые Visual C++ 6.0 элементы управления ActiveX по-прежнему хранятся в файлах, имеющих расширение `.OCX`.

Первоначально цель создания элементов управления VBX состояла в предоставлении программистам возможности включать в пользовательский интерфейс нестандартные элементы управления. Они позволяли без особых трудностей разработать элемент управления, который имел вид индикатора количества топлива или регулятора громкости. Однако почти сразу же программисты, работающие с VBX, от простых элементов управления перешли к модулям, включающим значительное количество вычислений и обработки. Точно так же многие элементы управления ActiveX являются на самом деле чем-то существенно большим, чем просто элементами управления. Они являются *компонентами*, которые могут быть использованы для быстрого построения мощных приложений.

На заметку

Если вы уже имеете опыт создания OCX в одной из более ранних версий Visual C++, у вас могло сложиться впечатление, что подобная работа является непростой. Однако пакет `Control Developer Kit`, интегрированный в новую версию Visual C++, принимает на себя большую часть работы по обеспечению требований ActiveX и дает вам возможность сконцентрироваться на вычислениях, отображении на экран или любых других действиях, для выполнения которых и предназначен данный элемент. Мастер `ActiveX Control` значительно упрощает работу, позволяя начать с уже имеющейся пустой заготовки.

Поскольку элементы управления являются небольшими серверами автоматизации ActiveX, они должны использоваться контроллерами автоматизации ActiveX. Чтобы не путать контроллер и элемент управления (что безотносительно к ActiveX есть одно и то же), вместо термина *контроллер автоматизации* будем пользоваться более привычным термином *приложение-контейнер* или просто *контейнер*. И Visual C++, и Visual Basic являются контейнерами, ими являются также многие программы, входящие в состав Office, и другие программы Microsoft.

В дополнение к методам и свойствам элементы управления ActiveX имеют дело с *событиями*. Говоря конкретнее, элемент управления *посылает* контейнеру сообщение о событии и делает это в том случае, когда происходит что-то, о чем следует уведомить контейнер. Например, когда пользователь делает щелчок в любом месте изображения, элемент обрабатывает щелчок (скажем, изменяет облик этого участка или выполняет какие-либо вычисления). Но ему необходимо, как правило, еще и уведомить о щелчке приложение-контейнер, чтобы оно выполнило, к примеру, открытие файла или какое-либо иное действие.

В этой главе вам был предоставлен краткий обзор концепций технологии ActiveX и используемой в ней терминологии, а также перечислены разнообразные возможности приложений, использующих элементы технологии ActiveX. Остальные главы этой части книги посвящены созданию приложений с помощью MFC и Мастеров Visual C++, поддерживающих технологию ActiveX.

ГЛАВА 14

Создание приложения-контейнера ActiveX

В этой главе...

Доработка приложения ShowString

Перемещение объекта и изменение его размеров

Выборка объектов и работа с несколькими объектами

Реализация в приложении технологии "перетащить и опустить"

Удаление объекта

Самый примитивный контейнер ActiveX можно создать, просто потребовав от мастера AppWizard сделать это, но возможности такого контейнера будут очень ограничены. Разобраться в том, как работает контейнер ActiveX и что следует предпринять для его реального использования, — куда более трудная задача. В этой главе на примере превращения приложения ShowString, разработанного в предыдущих главах, в контейнер ActiveX и последовательной его доработки до полнофункционального контейнера мы постараемся дать вам по возможности полное представление о функционировании ActiveX, так сказать, изнутри. Включение поддержки перетаскивания поднимет наше приложение до уровня современного проекта, имеющего интуитивный, документо-ориентированный пользовательский интерфейс. Прежде чем приступить к изучению дальнейшего материала, настоятельно рекомендуем вам познакомиться с содержанием предыдущей главы, если вы по каким-либо причинам не сделали этого ранее.

Доработка приложения ShowString

Первоначальная версия приложения ShowString была рассмотрена в главе 8. В ней не предусматривалась поддержка технологии ActiveX. Вы могли бы внести все требуемые изменения вручную, однако их общее число превышает 30. Проще и быстрее заново создать приложение ShowString — на этот раз с поддержкой функций контейнера ActiveX, — а затем внести в полученный текст программы изменения, необходимые для обеспечения расширенных функциональных возможностей приложения.

Генерация текста программы контейнера ActiveX с помощью мастера AppWizard

Создайте новое приложение ShowString в другом каталоге, выполнив в процессе настройки AppWizard почти те же установки, которые были сделаны при создании первой версии приложения ShowString в главе 8. Присвойте проекту имя ShowString, выберите вариант MDI-приложения, откажитесь от поддержки баз данных, потребуйте создания приложения как контейнера составных документов, выберите включение в приложение панели инструментов, строки состояния, режимов печати и предварительного просмотра документа, контекстно-зависимой справки и трехмерного дизайна элементов управления. И наконец, установите опции вставки комментариев в файлы исходных текстов и использования разделяемой библиотеки DLL.

На заметку

Хотя в настоящее время обсуждаемая технология называется ActiveX, в окнах настройки AppWizard используются ссылки на технологию поддержки составных документов. Кроме того, многие имена классов, которые используются в этой главе, содержат аббревиатуру OLE, а в комментариях имеются ссылки на технологию OLE. Хотя Microsoft уже изменила название этой технологии, соответствующие изменения в Visual C++ еще не проведены. Придется смириться с указанными противоречиями вплоть до выхода следующей версии продукта Visual C++.

Существует множество различий между приложением, которое вы только что создали, и "ничего не выполняющим" приложением, не поддерживающим функции контейнера ActiveX. Эти различия описываются и поясняются дальше в этом разделе, также анализируются вызываемые ими эффекты.

Меню

Появилось новое меню с идентификатором IDR_SHOWSTTYPE_CNTR_IP, показанное на рис. 14.1. В наименовании идентификатора содержится намек на контейнер, обладающий следующей особенностью: *содержащийся* (contained) в нем объект может быть отредактирован *на месте* (in place). Во время редактирования на месте панель меню будет состоять из *меню редактирования на месте* контейнера и *меню редактирования на месте* сервера. Две вертикальные полосы в центре меню IDR_SHOWSTTYPE_CNTR_IP являются разделителями — элементы меню сервера будут размещены между ними. Подробнее данный вопрос обсуждается в главе 15.

Меню Edit, показанное на рис. 14.2, включает четыре новые команды.

- Paste Special — эту команду следует выбирать для того, чтобы поместить в контейнер экземпляр объекта из буфера обмена (Clipboard).
- Insert New Object — при выборе этой команды на экране появится диалоговое окно Insert Object, показанное на рис. 14.3 и 14.4, пользуясь которым пользователь сможет поместить в контейнер экземпляр объекта.

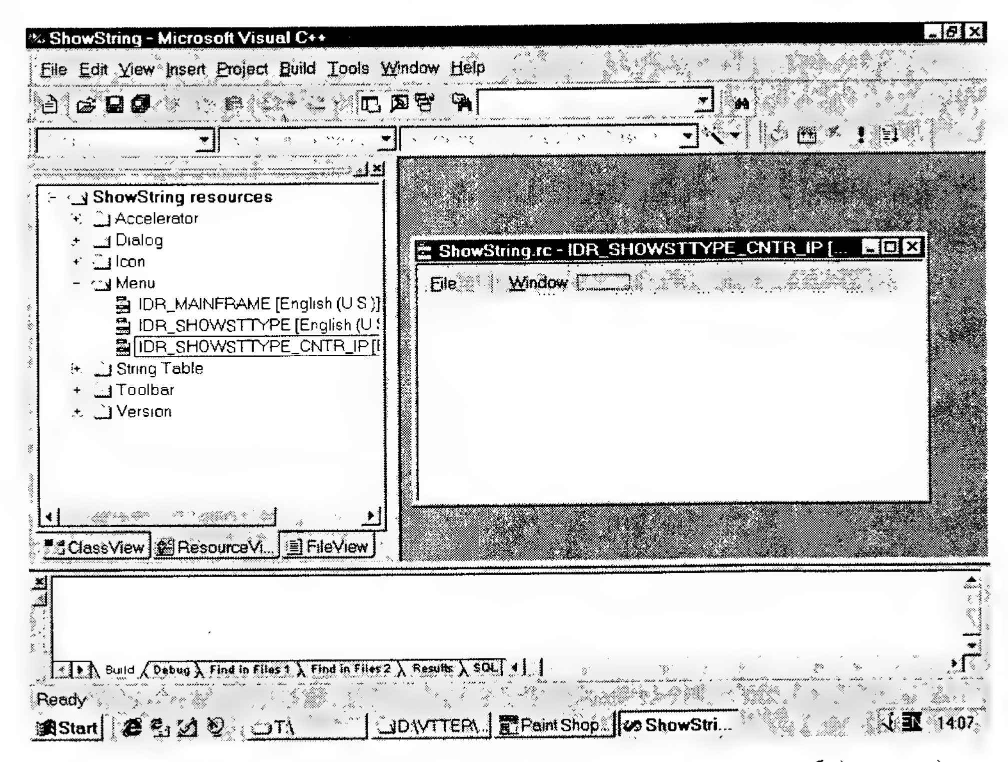


Рис. 14.1. Мастер AppWizard добавил к приложению новое меню, которое будет выводиться во время редактирования объекта на месте

- Links — это команда, выбор которой вызовет появление на экране диалогового окна Links, показанного на рис. 14.5; если объект уже связан с контейнером, это окно позволяет управлять способом обновления копии объекта после сохранения изменений в файле на диске.

Рис. 14.2. Мастер AppWizard добавил новые команды в меню Edit ресурса IDR_SHOWSTTYPE

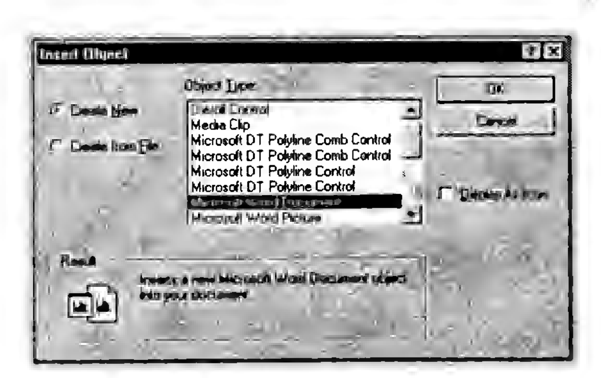


Рис. 14.3. Диалоговое окно Insert Object используется для внедрения новых объектов

- <<OLE VERBS GO HERE >>. Каждый тип объекта имеет различные связанные с ним команды, например Edit, Open или Play. Когда фокус перемещается на помещенный в контейнер экземпляр объекта, этот пункт меню заменяется соответственно типу объекта, выбранному из тех, которые присутствуют в диалоговом окне Insert Object. Для этого пункта формируется еще один уровень раскрывающегося меню, содержащий перечень команд, специфических для данного типа объектов, как показано на рис. 14.6.

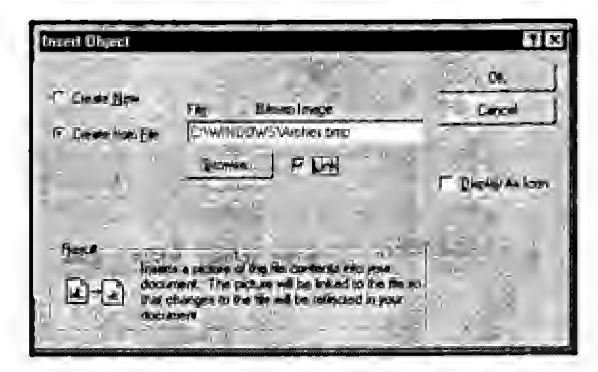


Рис. 14.4. Диалоговое окно Insert Object *используется для внедрения или связывания объекта, являющегося файлом*

Рис. 14.5. Диалоговое окно Links, *предоставляющее возможность определить метод обновления связанного объекта*

Класс CShowStringApp

По сравнению с методом `InitInstance()`, создаваемым мастером `AppWizard` для приложения, которое не являлось контейнером ActiveX, в функцию `CShowStringApp::InitInstance()` внесено несколько изменений. Фрагмент кода, показанный в листинге 14.1, выполняет инициализацию библиотек ActiveX (OLE).

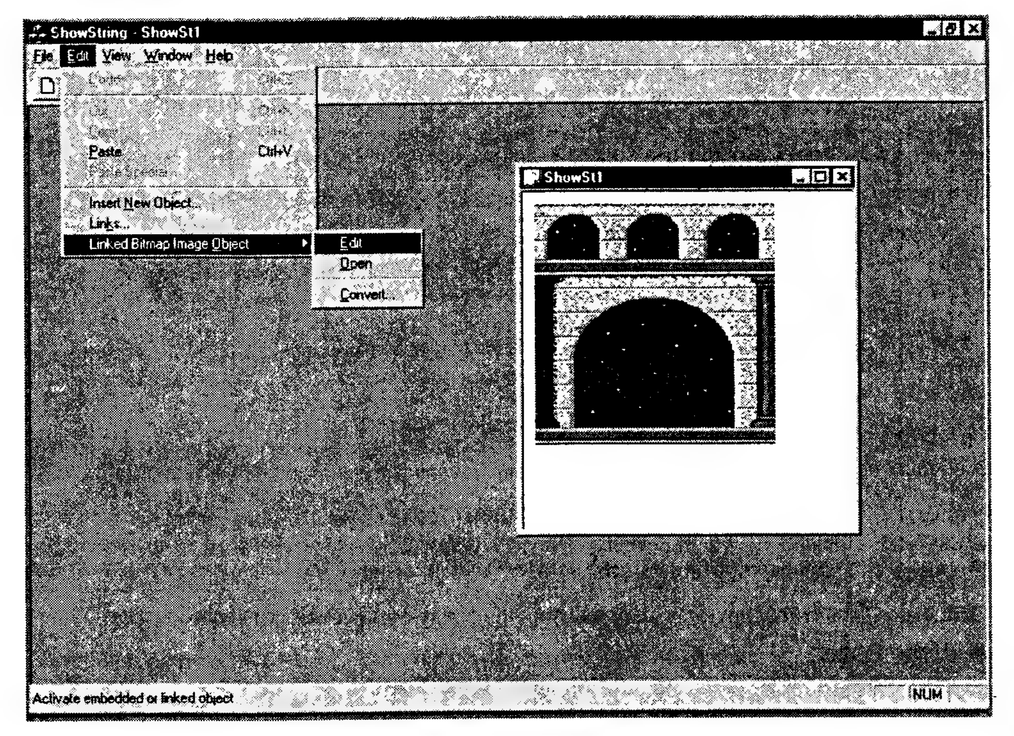


Рис. 14.6. Каждый тип объекта при получении фокуса помещает в меню Edit *дополнительное всплывающее меню, содержащее специфические для данного типа объекта команды*

Листинг 14.1. Фрагмент файла `ShowString.cpp` **— инициализация библиотек**

```
// Инициализация библиотек OLE.
if (!AfxOleInit())
   {
      AfxMessageBox(IDP_OLE_INIT_FAILED);
      return FALSE;
   }
```

Дополнительно в той же функции `CShowString::InitInstance()` после инициализации `MultyDocTemplate` и до вызова функции `AddDocTemplate()` вставлен оператор, добавляющий в реестр меню, которое будет использоваться при редактировании объектов на месте:

```
pDocTemplate->SetContainerInfo( IDR_SHOWSTTYPE_CNTR_IP);
```

Класс CShowStringDoc

Класс документа, `CShowStringDoc`, теперь наследуется от класса `COleDocumnt`, а не от класса `CDocument`, как прежде. Кроме того, в начало файла `ShowStringDoc.cpp` добавлена следующая директива:

```
#include "CntrItem.h"
```

Файл CntrItem.h содержит описание класса, экземпляр которого помещен в контейнер — CShowStringCntrItem. Этот класс будет рассмотрен ниже. Кроме того, в файле ShowStringDoc.cpp в карту сообщений были внесены изменения, показанные в листинге 14.2.

Листинг 14.2. Фрагмент файла ShowStringDoc.cpp — дополнительные элементы карты сообщений

```
    ON_UPDATE_COMMAND_UI(ID_EDIT_PASTE,
➥COleDocument::OnUpdatePasteMenu)
    ON_UPDATE_COMMAND_UI(ID_EDIT_PASTE_LINK,
➥COleDocument::OnUpdatePasteLinkMenu)
    ON_UPDATE_COMMAND_UI(ID_OLE_EDIT_CONVERT,
➥COleDocument::OnUpdateObjectVerbMenu)
    ON_COMMAND(ID_OLE_EDIT_CONVERT,
➥COleDocument::OnEditConvert)
    ON_UPDATE_COMMAND_UI(ID_OLE_EDIT_LINKS,
➥COleDocument::OnUpdateEditLinksMenu)
    ON_COMMAND(ID_OLE_EDIT_LINKS,
➥COleDocument::OnEditLinks)
    ON_UPDATE_COMMAND_UI(ID_OLE_VERB_FIRST,ID_OLE_VERB_LAST,
➥COleDocument::OnUpdateObjectVerbMenu)
```

Эти команды активизируют и деактивизируют следующие команды меню.

- Edit⇨Paste
- Edit⇨Paste Link
- Edit⇨Links
- Дополнительное меню команд OLE, включая команду Convert

Кроме того, новые макросы обрабатывают команды Convert и Edit⇨Links. Обратите внимание, что сообщения обрабатываются функциями, входящими в класс COleDocument, и вам не требуется создавать их самостоятельно.

В конструктор CShowStringDoc::CShowStringDoc() добавлена новая строка:

```
EnableCompoundFile();
```

Эта строка задействует обработку составных файлов. В функцию CShowStringDoc::Serialize() также добавлена строка:

```
COleDocument::Serialize(ar);
```

Этот вызов метода Serialize() базового класса обеспечит последовательную нумерацию всех помещенных в контейнер объектов.

Класс CShowStringView

В определении класса представления CShowStringView, как и в определении класса документа, добавлена строка подключения файла CntrItem.h. В карту сообщений класса представления добавлены следующие новые элементы:

```
ON_WM_SETFOCUS()
ON_WM_SIZE()
ON_COMMAND(ID_OLE_INSERT_NEW, OnInsertObject)
ON_COMMAND(ID_CANCEL_EDIT_CNTR, OnCancelEditCntr)
```

Эти строки определяют сообщения, которые будут перехватываться представлением дополнительно к тем сообщениям, которые оно перехватывало, не будучи контейнером ActiveX. Будут перехватываться сообщения VM_SETFOCUS, VM_SIZE, команда меню **Edit⇨Insert New Object** и отказ от редактирования, проводимого на месте. Акселератор для подключения этого сообщения к клавише <Esc> уже был определен ранее.

В файл ShowStringView.h добавлена новая переменная-член, как показано в листинге 14.3.

Листинг 14.3. Фрагмент файла ShowStringView.h — определение переменной m_pSelection

```
// m_pSelection предназначена для хранения выбора текущего
// CShowStringCntrItem. Во многих приложениях такая
// переменная-член не сможет адекватно представлять выборку,
// сделанную пользователем, например множественную выборку или
// выборку объекта, не являющегося объектом CShowStringCntrItem.
// Данный механизм выборки предоставляется только для
// того, чтобы помочь вам начать работу.

// TODO: замените этот механизм выборки другим,
// соответствующим характеру вашего приложения.

CShowStringCntrItem* m_pSelection;
```

Эта новая переменная-член присутствует и в конструкторе представления, показанном в листинге 14.4, а также в измененном методе OnDraw(), текст которого приведен в листинге 14.5.

Листинг 14.4. Файл ShowStringView.cpp — функция-конструктор

```
CShowStringView::CShowStringView()
{
   m_pSelection = NULL;
   // TODO: поместите сюда текст конструктора.
}
```

Листинг 14.5. Файл ShowStringView.cpp — функция CShowStringView::OnDraw()

```
void CShowStringView::OnDraw(CDC* pDC)
{
   CShowStringDoc* pDoc = GetDocument();
   ASSERT_VALID(pDoc);

   // TODO: добавьте сюда текст для вывода
   // на экран собственных данных приложения.

   // TODO: выведите на экран все экземпляры
   // объектов OLE, присутствующие в документе.

   // Вывод на экран осуществляется в произвольную позицию.
   // Данный программный текст следует удалить, когда будет
   // вставлен подготовленный вами реальный программный текст
   // прорисовки. Указанная позиция в точности соответствует
   // прямоугольнику, возвращаемому CShowStringCntrItem, что
   // и создает эффект редактирования на месте.

   // TODO: удалите этот программный текст, когда будет
   // завершено создание реального текста прорисовки.
```

```
    if (m_pSelection == NULL)
    {
        POSITION pos = pDoc->GetStartPosition();
        m_pSelection = (CShowStringCntrItem*) pDoc->GetNextClientItem(pos);
    }
    if (m_pSelection != NULL)
        m_pSelection->Draw(pDC, CRect(10, 10, 210, 210));
}
```

Исходный вариант функции OnDraw() выводит на экран только один экземпляр из объектов, помещенных в контейнер. Он не прорисовывает каких-либо данных самой программы, т.е. тех элементов ShowString, которые не содержат помещенных в них экземпляров объектов. В начале работы программы данные отсутствуют, но после ввода в приложение строки функция OnDraw() должна будет вывести ее на экран. Кроме того, исходный вариант этой функции выводит на экран только один из помещенных в контейнер объектов и к тому же выполняет вывод в фиксированный прямоугольник. По ходу работы над этой главой в функцию OnDraw() предстоит внести еще одно изменение.

Класс представления приобрел несколько следующих новых методов.

- OnInitialUpdate()
- IsSelected()
- OnInsertObject()
- OnSetFocus()
- OnSize()
- OnCancelEditCntr()

Каждая из этих функций будет обсуждаться в последующих разделах.

Метод OnInitialUpdate()

Метод OnInitialUpdate() вызывается перед самым первым выводом представления на экран. Автоматически сгенерированный текст этой функции (листинг 14.6) на удивление примитивен.

Листинг 14.6. Файл ShowStringView.cpp — функция CShowStringView::OnInitialUpdate()

```
void CShowStringView::OnInitialUpdate()
{
    CView::OnInitialUpdate();

    // TODO: удалите данный текст, когда будет
    // подготовлен реальный текст модели выборки.

    m_pSelection = NULL;    // Инициализация выборки.
}
```

Метод OnInitialUpdate() класса нашего приложения вызывает метод OnInitialUpdate() своего базового класса CView, в котором производится вызов функции Invalidate(), что приводит к полной перерисовке клиентской области окна.

Метод IsSelected()

Метод IsSelected() на данный момент не работает, поскольку существующий механизм выборки крайне примитивен. В листинге 14.7 показана созданная мастером заготовка этого метода. Позднее, когда мы займемся созданием реальной системы выборки, можно будет наглядно увидеть, как работает данный фрагмент программы.

Листинг 14.7. Файл ShowStringView.cpp — метод CShowStringView::IsSelected()

```
BOOL CShowStringView::IsSelected(const CObject* pDocItem) const
{
    // Предлагаемый ниже программный текст будет корректен
    // только в том случае, если выборка в программе может
    // состоять только из объектов CShowStringCntrItem.
    // При другом механизме выборки этот фрагмент должен
    // быть удален.

    // TODO: доработайте данную функцию, проверяющую
    // наличие выбранного клиентом экземпляра объекта OLE.

return pDocItem == m_pSelection;
}
```

Эта функция передает указатель экземпляру объекта в контейнере. Если данный указатель соответствует текущей выборке, возвращается TRUE.

Метод OnInsertObject()

Метод OnInsertObject() вызывается, когда пользователь выбирает команду **Edit⇨Insert New Object**. Эта функция довольно велика по размеру, и поэтому будет представлена вам по частям. Общая ее структура представлена в листинге 14.8.

Листинг 14.8. Файл ShowStringView.cpp — общая структура метода CShowStringView::OnInsertObject()

```
void CShowStringView::OnInsertObject()
{
    // Отображение диалогового окна Insert Object.

    CShowStringCntrItem* pItem = NULL;
    TRY
    {
        // Создание нового экземпляра объекта, включенного в этот документ.
        // Инициализация экземпляра объекта.
        // Задание выборки и обновление всех представлений.
    }
    CATCH(CException, e)
    {
        // Обработка неудачи при попытке создания.
    }
    END_CATCH
    // Завершение обработки.
}
```

В данном листинге каждый комментарий замещает некоторую часть текста функции (они будут обсуждаться в последующей части данного раздела). Макросы TRY и CATCH, кстати, являются устаревшими формами обработки исключений, которые обсуждаются в главе 26.

Прежде всего, данная функция выводит на экран диалоговое окно Insert Object, как показано в листинге 14.9.

Листинг 14.9. Файл ShowStringView.cpp — фрагмент метода CShowStringView::OnInsertObject(); отображение диалогового окна Insert Object

```
// Вывод стандартного диалогового окна Insert Object для
// получения информации о новом объекте CShowStringCntrItem.
COleInsertDialog dlg;
if (dlg.DoModal() != IDOK)
   return;
BeginWaitCursor();
```

Если пользователь сделает щелчок на кнопке Cancel, эта функция возвратит управление и ничего вставлено не будет. Если пользователь щелкнет на кнопке OK, указатель мыши примет вид песочных часов и будет оставаться таким вплоть до окончания процесса обработки.

Для создания нового экземпляра объекта в функцию включены операторы, приведенные в листинге 14.10.

Листинг 14.10. Файл ShowStringView.cpp — фрагмент метода CShowStringView::OnInsertObject(); создание нового элемента

```
// Создание нового экземпляра объекта, включенного в данный документ.
CShowStringDoc* pDoc = GetDocument();
ASSERT_VALID(pDoc);
pItem = new CShowStringCntrItem(pDoc);
ASSERT_VALID(pItem);
```

В этом фрагменте программы прежде всего запрашивается подтверждение существования документа. Это выполняется несмотря на то, что команда меню активизируется только в случае, когда существует хотя бы один документ. Затем создается новый экземпляр класса контейнера, и ему передается указатель на документ. Как вы узнаете из раздела, посвященного объекту CShowStringCntrItem, экземпляр элемента контейнера содержит указатель на документ, в который он включен.

Фрагмент текста программы, приведенный в листинге 14.11, предназначен для инициализации этого экземпляра объекта.

Листинг 14.11. Файл ShowStringView.cpp — фрагмент метода CShowStringView::OnInsertObject(); инициализация вставленного экземпляра объекта

```
// Инициализация экземпляра объекта данными, полученными из диалога.
if (!dlg.CreateItem(pItem))
   AfxThrowMemoryException(); // Обработка исключений.
ASSERT_VALID(pItem);
// Если экземпляр объекта создан из списка классов (не из файла),
// то запускается сервер для редактирования этого экземпляра.
if (dlg.GetSelectionType() == COleInsertDialog::createNewItem)
   pItem->DoVerb(OLEIVERB_SHOW, this);

ASSERT_VALID(pItem);
```

В фрагменте текста программы, приведенного в листинге 14.11, вызывается функция CreateItem() класса диалога COleInsertDialog. Может показаться странным, что подобная функция входит в состав указанного класса, но этой функции необходимо знать ответы пользователя на все вопросы, заданные в диалоговом окне. Поэтому, если бы эта функция принадлежала другому классу, ей пришлось бы запрашивать диалоговое окно о типе и имени файла,

выяснять, что необходимо выполнить (связывание или внедрение), и т.д. Данная функция вызывает методы класса, которому принадлежит экземпляр объекта контейнера, такие как CreateLinkFromFile(), CreateFromFile(), CreateNewItem() и т.д. Поэтому дело даже не в том, что подпрограмма для реального заполнения объекта данными из файла принадлежит диалоговому окну, а скорее в том, что вместо взаимного обмена информацией между объектами работа просто разделяется между ними.

Далее у диалогового окна запрашивается, был ли это новый экземпляр объекта. Если да, то вызывается сервер для его редактирования. Объекты, создаваемые из файла, могут отображаться непосредственно.

И наконец, обновляются выборка и представления, как показано в листинге 14.12.

Листинг 14.12. Файл ShowStringView.cpp — фрагмент метода CShowStringView::OnInsertObject(); обновление выборки и представлений

```
// Поскольку допустим произвольный пользовательский интерфейс,
// здесь выбирается последний вставленный экземпляр объекта.

// TODO: замените процедуру установки выборки процедурой,
// которая соответствует вашему приложению.

m_pSelection = pItem;  // Устанавливает выборку на последний
                       // вставленный экземпляр объекта.
pDoc->UpdateAllViews(NULL);
```

Если создание объекта завершится неудачей, выполнение закончится в блоке CATCH, показанном в листинге 14.13.

Листинг 14.13. Файл ShowStringView.cpp — блок CATCH в методе CShowStringView::OnInsertObject()

```
CATCH(CException, e)
{
   if (pItem != NULL)
   {
      ASSERT_VALID(pItem);
      pItem->Delete();
   }
   AfxMessageBox(IDP_FAILED_TO_CREATE);
}
END_CATCH
```

В этом блоке удаляется созданный экземпляр объекта и выводится окно сообщения.

Наступил момент, когда указатель мыши в виде песочных часов можно убрать с экрана:

```
EndWaitCursor();
```

Метод OnSetFocus()

Метод OnSetFocus(), текст которого приведен в листинге 14.14, вызывается всякий раз, когда фокус перемещается на данное представление.

Листинг 14.14. Файл ShowStringView.cpp — метод CShowStringView::OnSetFocus()

```
void CShowStringView::OnSetFocus(CWnd* pOldWnd)
{
   COleClientItem* pActiveItem = GetDocument()->GetInPlaceActiveItem(this);
   if (pActiveItem != NULL &&
```

```
        pActiveItem->GetItemState() == COleClientItem::activeUIState)
    {
        // Необходимо поместить фокус на данный экземпляр
        // объекта, если он находится в этом же представлении.
        CWnd* pWnd = pActiveItem->GetInPlaceWindow();
        if (pWnd != NULL)
        {
            pWnd->SetFocus();  // Не вызывайте базовый класс.
            return;
        }
    }
    CView::OnSetFocus(pOldWnd);
}
```

Если существует активный экземпляр объекта и его сервер загружен, то фокус перемещается на этот активный экземпляр объекта. Если подобного объекта нет, фокус остается на прежнем окне, а для пользователя это выглядит так, как будто его щелчок был проигнорирован.

Метод OnSize()

Метод OnSize(), текст которого приведен в листинге 14.15, вызывается, когда пользователь изменяет размеры окна приложения.

Листинг 14.15. Файл ShowStringView.cpp — метод CShowStringView::OnSize()

```
void CShowStringView::OnSize(UINT nType, int cx, int cy)
{
    CView::OnSize(nType, cx, cy);
    COleClientItem* pActiveItem = GetDocument()->GetInPlaceActiveItem( this);
    if (pActiveItem != NULL)
        pActiveItem->SetItemRects();
}
```

Эта функция, используя метод базового класса, изменяет размер представления, а затем, если присутствует активный экземпляр объекта, сообщает ему о необходимости привести изображение в соответствие с новыми размерами представления.

Метод OnCancelEditCntr()

Метод OnCancelEditCntr() вызывается в том случае, если пользователь, выполнявший редактирование внедренного объекта на месте, нажал клавишу <Esc>. Сервер этого объекта должен быть закрыт, а сам объект деактивизирован. Текст метода показан в листинге 14.16.

Листинг 14.16. Файл ShowStringView.cpp — метод CShowStringView::OnCancelEditCntr()

```
void CShowStringView::OnCancelEditCntr()
{

    // Закрывается любой редактируемый на месте экземпляр
    // объекта в этом представлении.
    COleClientItem* pActiveItem =
        GetDocument()->GetInPlaceActiveItem(this);
    if (pActiveItem != NULL)
    {
        pActiveItem->Close();
    }
    ASSERT(GetDocument()->GetInPlaceActiveItem(this) == NULL);
}
```

Класс CShowStringCntrItem

В приложение ShowString добавлен совершенно новый класс контейнерного объекта. Он описывает объект, помещенный в документ. Как вы могли уже видеть, и документ, и представление обращаются к объектам этого класса достаточно часто (прежде всего, при работе с переменной-членом m_pSelection класса CShowStringView). Класс CShowStringCntrItem не имеете переменных-членов, кроме тех, которые он наследует от базового класса COleClientItem. Однако в нем переопределяется множество методов, а именно — следующие.

- Конструктор
- Деструктор
- GetDocument()
- GetActiveView()
- OnChange()
- OnActivate()
- OnGetItemPosition()
- OnDeactivateUI()
- OnChangeItemPosition()
- AssertValid()
- Dump()
- Serialize()

Конструктор класса просто передает указатель на документ базовому классу. Деструктор вообще ничего не делает. Функции GetDocument() и GetActiveView() являются передаточными функциями, которые возвращают наследованные от базового класса переменные-члены посредством вызова функций базового класса с тем же самым именем и передачи на выход полученного результата.

Функция OnChange() является первой из всех указанных в приведенном выше списке функций, которая содержит более одной строки программного кода (листинг 14.17).

Листинг 14.17. Файл CntrItem.cpp — метод CShowStringCntrItem::OnChange()

```
void CShowStringCntrItem::OnChange(OLE_NOTIFICATION nCode,
   DWORD dwParam)
{
   ASSERT_VALID(this);

   COleClientItem::OnChange(nCode, dwParam);

   // Когда экземпляр объекта редактируется (на месте или в
   // отдельном окне), он посылает уведомления OnChange,
   // указывающие на изменение его состояния или внешнего
   // вида его содержимого.

   // TODO: потребовать перерисовки экземпляра объекта
   // посредством вызова метода UpdateAllViews() (с учетом
   // всех особенностей вашего приложения).

   GetDocument()->UpdateAllViews(NULL);
      // Этот вариант просто обновляет ВСЕ представления
      // без учета их особенностей.
}
```

Действительно, в этой функции присутствует целых три оператора. А имеющиеся комментарии оказываются более полезными, чем сами операторы. Когда пользователь изменяет объект, помещенный в контейнер, сервер уведомляет об этом контейнер. Вызов функции UpdateAllViews() является в данном случае довольно грубым методом обновления экрана, но требуемая цель, безусловно, будет достигнута.

Функция OnActivate() (листинг 14.18) вызывается, когда пользователь выполняет двойной щелчок на экземпляре объекта для его активизации и редактирования на месте. Как правило, объекты ActiveX имеют тип активизации *снаружи вовнутрь* (outside-in). Это означает, что одиночный щелчок на них лишь выбирает их, но не активизирует. Для активизации такого объекта необходимо сделать на нем двойной щелчок либо выполнить одиночный щелчок, а затем выбрать соответствующую команду OLE в меню Edit.

Листинг 14.18. Файл CntrItem.cpp — метод CShowStringCntrItem::OnActivate()

```
void CShowStringCntrItem::OnActivate()
{
   // В каждом окне допускается только один активизированный
   // на месте экземпляр объекта.
   CShowStringView* pView = GetActiveView();
   ASSERT_VALID(pView);
   COleClientItem* pItem = GetDocument()->GetInPlaceActiveItem(pView);
   if (pItem != NULL && pItem != this)
      pItem->Close();

   COleClientItem::OnActivate();
}
```

В подпрограмме прежде всего выполняется проверка корректности текущего представления; далее закрываются все активные экземпляры объектов, если таковые имеются, а затем активизируется данный объект.

Функция OnGetItemPosition() (листинг 14.19) вызывается как часть процесса активизации объекта на месте.

Листинг 14.19. Файл CntrItem.cpp — метод CShowStringCntrItem::OnGetItemPosition()

```
void CShowStringCntrItem::OnGetItemPosition(CRect& rPosition)
{
   ASSERT_VALID(this);

   // В процессе активизации на месте функция
   // CShowStringCntrItem::OnGetItemPosition будет вызываться
   // для определения расположения данного экземпляра объекта.
   // Программный код, подставляемый в эту функцию с помощью
   // мастера AppWizard, просто возвращает координаты жестко
   // заданного прямоугольника. Как правило, этот прямоугольник
   // должен отражать текущую позицию экземпляра объекта по
   // отношению к представлению, используемому для активизации.
   // Можно определить параметры представления, вызвав
   // функцию CShowStringCntrItem::GetActiveView.


   // TODO: обеспечить возвращение в переменной rPosition
   // координат (в пикселях) корректного прямоугольника.

   rPosition.SetRect(10, 10, 210, 210);
}
```

Как и в случае функции OnChange(), помещенные в текст комментарии оказываются полезнее самого исходного текста функции. На данный момент функция класса представления OnDraw() рисует помещенный в контейнер объект в жестко заданном прямоугольнике. Соот-

ветственно и наша функция возвращает координаты того же самого прямоугольника. Ну а вам оставлены инструкции, чтобы написать реальную подпрограмму, которая будет запрашивать у активного представления, где же расположить данный объект.

Функция OnDeactivateUI() (листинг 14.20) вызывается в том случае, когда объект из активного состояния переводится в неактивное.

Листинг 14.20. Файл CntrItem.cpp — метод CShowStringCntrItem::OnDeactivateUI()

```
void CShowStringCntrItem::OnDeactivateUI(BOOL bUndoable)
{
   COleClientItem::OnDeactivateUI(bUndoable);

   // Убирает объект с экрана, если он не является
   // объектом с активизацией снаружи вовнутрь.
   DWORD dwMisc = 0;
   m_lpObject->GetMiscStatus(GetDrawAspect(), &dwMisc);
   if (dwMisc & OLEMISC_INSIDEOUT)
      DoVerb(OLEIVERB_HIDE, NULL);
}
```

Хотя по умолчанию для объекта, помещенного в контейнер, устанавливается тип активизации снаружи вовнутрь (о чем шла речь чуть выше), можно использовать и объекты с активизацией *изнутри наружу* (inside-out). Такие объекты активизируются сразу же при помещении на них указателя мыши, а щелчок на подобном объекте имеет такой же эффект, как и щелчок в данной области при редактировании объекта. Например, если помещенный в контейнер объект является электронной таблицей, щелчок на ней повлечет за собой выбор ячейки, в которой он был выполнен. Это, безусловно, весьма удобно для пользователя, который в подобном случае может полностью игнорировать границы между контейнером и помещенным в него экземпляром объекта. Однако разработка соответствующих программ является очень непростым делом.

Функция OnChangeItemPosition() вызывается в том случае, когда объект перемещается в процессе редактирования на месте. Она также содержит по большей части комментарии, как показано в листинге 14.21.

Листинг 14.21. Файл CntrItem.cpp — метод CShowSringCntrItem::OnChangeItemPosition()

```
BOOL CShowStringCntrItem::OnChangeItemPosition(const CRect& rectPos)
{
   ASSERT_VALID(this);

   // В процессе активизации на месте функция
   // CShowStringCntrItem::OnChangeItemPosition вызывается
   // сервером для изменения расположения окна редактирования
   // на месте. Обычно это происходит при таком изменении
   // данных в обрабатываемом сервером документе, которое
   // влечет за собой изменение размеров окна либо является
   // результатом прямого изменения пользователем размеров
   // окна редактирования на месте.
   //
   // По умолчанию здесь необходимо выполнить обращение к
   // базовому классу, который осуществит вызов функции
   // COleClientItem::SetItemRects для перемещения
   // объекта в новую позицию.

   if (!COleClientItem::OnChangeItemPosition(rectPos))
```

```
        return FALSE;

    // TODO: обновить любое кэшированное изображение
    // прямоугольника/участка экземпляра объекта.
    return TRUE;
}
```

Предполагается, что данный фрагмент программы будет управлять перемещением объекта, но на самом деле этого не происходит по той простой причине, что функция OnDraw() всегда рисует экземпляр объекта в контейнере на одном и том же месте.

Функции AssertValid() и Dump() являются отладочными функциями, которые просто вызывают функции базового класса. Последней из функций класса CShowStringCntrItem является функция Serialize(), которая вызывает функцию COleDocument::Serialize(), которая, в свою очередь, вызывает функцию Serialize() документа, как мы уже видели ранее. Текст последней функции обсуждаемого класса приведен в листинге 14.22.

Листинг 14.22. Файл CntrItem.cpp — метод CShowStringCntrItem::Serialize()

```
void CShowStringCntrItem::Serialize(CArchive& ar)
{
    ASSERT_VALID(this);

    // Прежде чем читать данные класса COleClientItem, вызовите
    // базовый класс. Поскольку это действие требует установки
    // указателя m_pDocument, возращаемого функцией
    // CShowStringCntrItem::GetDocument, желательно
    // предварительно вызвать метод Serialize базового класса.

    COleClientItem::Serialize(ar);

    // Сохранение-восстановление данных, специфических для
    // функции CShowStringCntrItem.

    if (ar.IsStoring())
    {
        // TODO: поместите сюда операторы для сохранения.
    }
    else
    {
        // TODO: поместите сюда операторы для считывания.
    }
}
```

Все, что на данный момент выполняет этот фрагмент программы, — вызывает функцию базового класса. Метод COleDocument::Serialize()для отслеживания обработки различных объектов, помещенных в контейнер, запоминает или загружает некоторое количество счетчиков и других переменных. Затем для реальной обработки объекта вызываются вспомогательные функции, такие как WriteItem() или ReadItem(). Работа этих функций и вспомогательных функций, которые они вызывают, для большинства остается абсолютно скрытой. Если у вас появится желание познакомиться с ними поближе, найдите их тексты в папке с исходными текстами MFC (C:\MSDEV\MFC\SRC в большинстве случаев) в файле olecli1.cpp. Эти функции успешно справляются со стоящей перед ними задачей, обеспечивая упорядочение помещаемых в контейнер объектов.

Недостатки данного контейнера

На данной стадии разработки приложение-контейнер, безусловно, еще не является полноценным приложением ShowString. А причина этого очень проста — нужно выполнить все, что указано в комментариях // TODO (ЧТО СДЕЛАТЬ). Тем не менее то, что мы получили действующий контейнер, хорошо демонстрирует достоинства классов MFC COleDocument и COleClientItem. А раз так, отчего бы нам не построить его и не опробовать в работе? После того как трансляция будет закончена и приложение запущено, выберите в его окне команду Edit⇨Insert New Object и поместите в него какой-либо графический файл типа .bmp. Сейчас, когда вы уже знакомы с текстами программ, у вас не должно вызвать удивления, что после двойного щелчка на объекте немедленно запустится редактор Paint для редактирования картинки на месте (рис. 14.7).

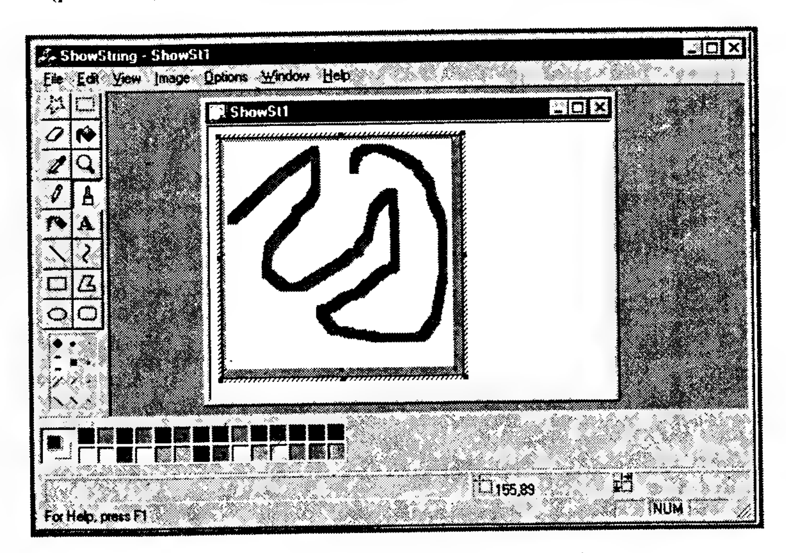


Рис. 14.7. Созданная заготовка приложения-контейнера может реально помещать объект в контейнер и активизировать его для редактирования на месте — этот графический объект редактируется в Paint

Для отмены выбора данного объекта и возврата управления контейнеру щелкните где-либо вне графического объекта. Однако вы увидите, что ничего не произошло. Щелкните вне документа, и снова ничего не произойдет. Вероятно, у вас возникнет сомнение: "Работаю ли я с ShowString?". Выберите команду File⇨New, и вы увидите, что работа с ShowString продолжается. Меню и панели инструментов Paint исчезнут с экрана и будет создан новый документ ShowString. Вновь сделайте щелчок на графическом объекте и вновь для его редактирования будет запущен Paint. Как можно вставить еще один объект в первый документ, если на экране отображаются меню и панели инструментов Paint? Для завершения редактирования на месте нажмите клавишу <Esc>, и на экране вновь появятся меню приложения ShowString. Поместите в контейнер диаграмму Excel. В результате, как только будет вставлена диаграмма Excel, графический объект исчезнет, что и показано на рис. 14.8. К сожалению, данный контейнер оставляет желать лучшего.

Для отмены редактирования на месте нажмите клавишу <Esc>, и вы увидите, что изображение объекта на экране уменьшилось (рис. 14.9). Произошло это по той причине, что ShowString отображает все помещенные в контейнер экземпляры объектов в прямоугольнике

размером 200×200 пикселей, поэтому, чтобы уместиться в заданные размеры, диаграмму пришлось несколько сжать. Принятие решения о том, как разместить объект в пространстве, предоставленном ему контейнером, возлагается на сервер (в данном случае — на Excel).

Как видите, для превращения имеющейся заготовки в полноценный контейнер еще предстоит сделать очень многое. Но прежде всего давайте вернем ей функциональные возможности самого приложения ShowString.

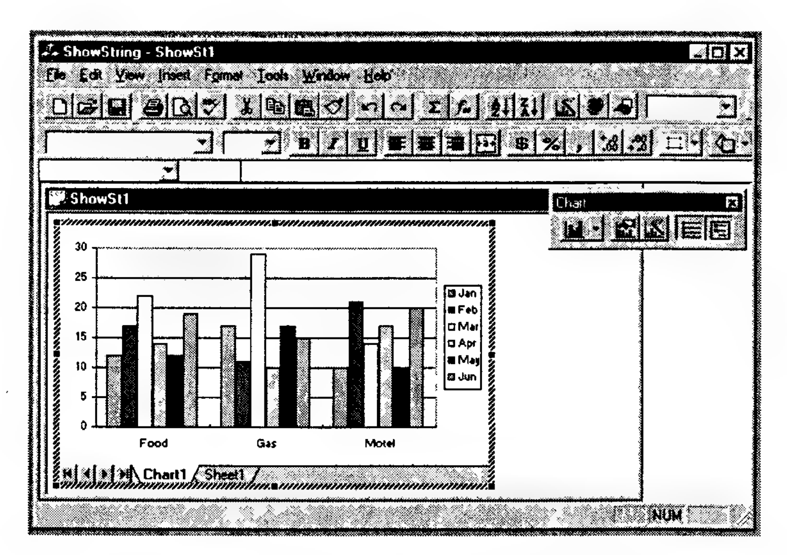


Рис. 14.8. Вставка диаграммы Excel будет успешной, но при этом из документа исчезнет графический объект

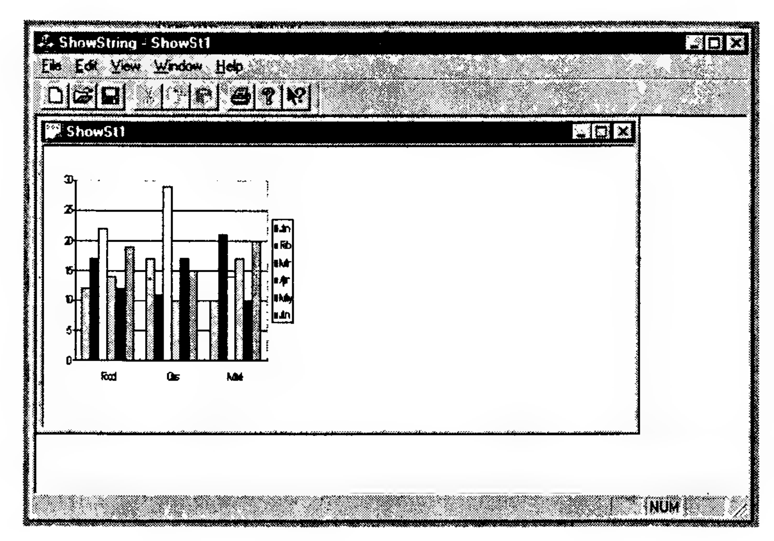


Рис. 14.9. Когда экземпляр объекта не активен, он выглядит на экране несколько по-другому

Восстановление функциональных возможностей ShowString

В этом разделе мы вкратце вспомним все операции, которые были выполнены в главе 8. Приступая к работе, откройте файлы прежнего приложеиия ShowString — это позволит вам везде, где только возможно, скопировать тексты программ и ресурсы в новую программу. Выполните следующие операции.

1. В файле ShowStringDoc.h добавьте в определение класса закрытые переменные-члены и открытую функцию Get().
2. В метод CShowStringDoc::Serialize() вставьте операторы для сохранения и восстановления определенных выше переменных. Вызов метода COleDocument:: Serialize() оставьте на прежнем месте.
3. В метод CShowStringDoc::OnNewDocument() вставьте операторы инициализации переменных-членов.
4. В метод CShowStringDoc::OnDraw() перед операторами обработки помещенных в контейнер объектов добавьте программный текст для отображения строки на экране. Удалите комментарии TODO, относящиеся к отображению собственных данных приложения.
5. Скопируйте меню Tools в новое приложение-контейнер из старого ShowString. Для открытия старого файла ShowString.rc выберите команду File⇨Open, раскройте меню IDR_SHOWSTTYPE, сделайте щелчок на меню Tools и выберите команду Edit⇨Copy. Раскройте меню IDR_SHOWSTTYPE нового приложения ShowString, сделайте щелчок на меню Window и выберите команду Edit⇨Paste. Будьте внимательны — не сделайте вставки в меню IDR_SHOWSTTYPE_CNTR_IP.
6. Добавьте акселератор <Ctrl+T> для команды ID_TOOLS_OPTIONS. Добавление выполните только для акселератора IDR_MAINFRAME.
7. Удалите из нового ShowString диалоговое окно IDD_ABOUTBOX. Скопируйте из старого ShowString в новое ресурсы IDD_ABOUTBOX и IDD_OPTIONS.
8. Пока ресурс IDD_OPTIONS сохраняет на себе фокус, выберите команду View⇨Class Wizard. Создайте класс COptionsDialog, аналогичный тому, который присутствовал в исходном приложении ShowString.
9. С помощью ClassWizard свяжите элементы управления диалогового окна с переменными-членами класса COptionDialog.
10. Используйте ClassWizard для организации перехвата команды ID_TOOLS_OPTIONS классом CShowStringDoc.
11. В файле ShowStringDoc.cpp замените созданную ClassWizard версию функции CShowStringDoc::OnToolsOptions() версией этой функции из прежнего ShowString, выводящей диалоговое окно.
12. Добавьте в файл ShowStringDoc.cpp директиву #include "OptionsDialog.h", поместив ее после присутствующих в файле директив #inlude.

Выполните для приложения команду Build, исправляя в ходе компиляции все возможные мелкие ошибки, а затем запустите его на выполнеиие. Приложение должно работать в точности так, как прежде, выводя в центре окна представления строку Hello, world!. Убедитесь, что диалоговое окно Options по-прежнему работает и что вы восстановили все функции, присущие прежнему приложению ShowString. Затем увеличьте насколько возможно размеры окна приложения, чтобы помещенный в него объект не перекрывался строкой. Вставьте в документ диаграмму Excel, как мы делали это выше, и нажмите клавишу <Esc> для отмены режима редактирования на месте. Работа выполнена: у вас есть приложение ShowString, которое приобрело функциональные возможности контейнера ActiveX.

Перемещение объекта и изменение его размеров

Хотя в созданный нами контейнер ActiveX пока можно поместить лишь один экземпляр объекта, первое, что мы сделаем, расширяя возможности ShowString, — предоставим пользователю возможность перемещать этот объект и изменять его размеры. Облегчить работу пользователя можно, обеспечив вывод на экран перемещаемого объекта *трассирующей рамки*, имеющей вид прямоугольника, нарисованного пунктирной линией. Последняя задача легко решается с помощью класса MFC CRectTracker.

Сначала добавим переменную-член в класс помещенного в контейнер объекта (CShowStringCntrItem), определяемый в файле CntrItem.h. Эта переменная предназначена для хранения координат прямоугольника, занимаемого в окне приложения объектом, помещенным в контейнер. На вкладке ClassView сделайте щелчок правой кнопкой мыши на имени класса CShowStringCntrItem и выберите в открывшемся контекстном меню команду Add Member Variable. Определите для переменной тип CRect, имя m_rect и тип доступа public.

Переменную m_rect необходимо инициализировать один-единственный раз в функции, вызываемой при первом использовании помещенного в контейнер объекта. Хотя классы представления содержат метод OnInitialUpdate(), а классы документа — метод OnNewDocument(), классы помещенного в контейнер объекта не содержат никаких подобных однократно вызываемых методов, кроме конструктора. Поэтому инициализация прямоугольника будет выполняться в конструкторе, как показано в листинге 14.23.

Листинг 14.23. Файл CntrItem.cpp — конструктор

```
CShowStringCntrItem::CShowStringCntrItem(CShowStringDoc* pContainer)
   : COleClientItem(pContainer)
{
   m_rect = CRect(10,10,210,210);
}
```

Параметры прямоугольника те же, которые AppWizard помещает в создаваемую им заготовку метода OnDraw(). Теперь следует присвоить переменной-члену m_rect значение и использовать ее. Методы, в которые необходимо внести изменения, представляются в том же порядке, что и в предыдущем разделе при рассмотрении объекта класса CShowStringView.

Сначала в методе CShowStringView::OnDraw() найдите следующий оператор:

```
m_pSelection->Draw(pDC, CRect(10, 10, 210, 210));
```

Замените его:

```
m_pSelection->Draw(pDC, m_pSelection->m_rect);
```

Затем измените метод CShowStringCntrItem::OnGetItemPosition(), который должен возвращать данный прямоугольник. Удалите все комментарии и оператор установки жестко заданных параметров прямоугольника (оставьте только вызов макрокоманды ASSERT_VALID) и добавьте такую строку:

```
rPosition = m_rect;
```

Родственная функция CShowStringCntrItem::OnChangeItemPosition() вызывается, когда пользователь перемещает объект. Именно здесь переменная m_rect получает новые значения, отличные от исходных. Удалите комментарии и сразу после вызова функции базового класса COleClientItem::OnChangeItemPosition() вставьте следующие операторы:

```
m_rect = rectPos;
GetDocument()->SetModifiedFlag();
GetDocument()->UpdateAllViews(NULL);
```

И наконец, новая переменная-член должна быть использована в методе CShowStringCntrItem::Serialize(). Удалите все комментарии и добавьте новые строки в блоки сохранения и восстановления так, чтобы код метода приобрел вид, показанный в листинге 14.24.

Листинг 14.24. Файл CntrItem.cpp — метод CShowStringCntrItem::Serialize()

```
void CShowStringCntrItem::Serialize(CArchive& ar)
{
   ASSERT_VALID(this);
   // Прежде чем читать данные класса COleClientItem, вызовите
   // базовый класс. Поскольку это действие требует установки
   // указателя m_pDocument, возращаемого функцией
   // CShowStringCntrItem::GetDocument, полезно будет
   // предварительно вызвать метод Serialize базового класса.

   COleClientItem::Serialize(ar);

   // Теперь сохраним-восстановим данные, специфические для
   // функции CShowStringCntrItem.

   if (ar.IsStoring())
   {
      ar << m_rect;
   }
   else
   {
      ar >> m_rect;
   }
}
```

Выполните для приложения команду Build и запустите его. Вставьте в документ графический объект типа .bmp и нарисуйте что-нибудь. Для окончания редактирования на месте нажмите клавишу <Esc>, и то, что вы нарисовали, появится в верхнем правом углу окна приложения, рядом со строкой Hello, world!. Выберите команду Edit⇨Bitmap Image Object⇨Edit. (Команда Open дает возможность редактировать объект в отдельном окне.) Воспользуйтесь появившимися размерными маркерами для изменения размера рисунка, а затем перетащите его влево. Нажав клавишу <Esc>, завершите редактирование на месте. Рисунок останется в новой позиции окна, что нам и требовалось.

Теперь займемся трассирующей рамкой. Справочное пособие Microsoft рекомендует для реализации этой возможности написать вспомогательную функцию SetupTracker(). Добавьте следующие строки в функцию CShowStringView::OnDraw(), сразу после вызова m_pSelection->Draw():

```
CRectTracker trackrect;
SetupTracker(m_pSelection,&trackrect);
trackrect.Draw(pDC);
```

Внимание!

Однострочный оператор, следовавший за оператором if, не был заключен в фигурные скобки — не забудьте вставить их. Полностью оператор if теперь должен выглядеть так.

```
if (m_pSelection !=NULL)
{
  m_pSelection->Draw(pDC, m_pSelection->m_rect);
  CRectTracker trackrect;
  SetupTracker(m_pSelection,&trackrect);
  trackrect.Draw(pDC);
}
```

В файле ShowStringView.h в определение класса добавьте следующий открытый метод:

```
void SetupTracker(CShowStringCntrItem* item,
    CRectTracker* track);
```

В файл ShowStringView.cpp, сразу же после определения деструктора, поместите текст программы, приведенный в листинге 14.25.

Листинг 14.25. Файл ShowStringView.cpp — метод CShowStringView::SetupTracker()

```
void CShowStringView::SetupTracker(CShowStringCntrItem* item,
    CRectTracker* track)
{
    track->m_rect = item->m_rect;

    if (item == m_pSelection)
    {
        track->m_nStyle |= CRectTracker::resizeInside;
    }

    if (item->GetType() == OT_LINK)
    {
        track->m_nStyle |= CRectTracker::dottedLine;
    }
    else
    {
        track->m_nStyle |= CRectTracker::solidLine;
    }
    if (item->GetItemState() == COleClientItem::openState ||
        item->GetItemState() == COleClientItem::activeUIState)
    {
        track->m_nStyle |= CRectTracker::hatchInside;
    }
}
```

Сначала в этом фрагменте программы определяется прямоугольник рамки объекта, помещенного в контейнер, а затем на него накладывается стиль. Существуют следующие типы стилей:

- solidLine (сплошная линия) — используется для внедренных объектов;
- dottedLine (точечная линия) — используется для связанных объектов;
- hatchedBorder (рамка с косой штриховкой) — используется для объектов, активизированных на месте;
- resizeInside (маркеры размера, размещенные внутри рамки) — используются для выбранных объектов;
- resizeOutside (маркеры размера, размещенные снаружи рамки) — используются для выбранных объектов;

- hatchInside (косая штриховка внутри рамки) — используется для объектов, сервер которых запущен.

При этом сравниваются указатели на данный объект и на текущую выборку. Если они совпадают, данный объект является выбранным и для него выводятся размерные маркеры. Вам самим следует решить, как вывести эти маркеры — внутри или снаружи рамки объекта. Затем объект анализируется: определяется, является он связанным (точечная линия) или не является (сплошная линия). И наконец, к активному объекту добавляется штриховка.

Выполните для приложения команду Build, а затем запустите его на выполнение. Опробуйте, как оно теперь работает. Как и прежде, у вас нет возможности начать редактирование помещенного в контейнер экземпляра объекта, сделав на нем двойной щелчок, — воспользуйтесь командой Edit выпадающего меню, добавленного в конец основного меню Edit программы. Невозможно выполнить перемещение неактивного объекта или изменение его размеров, но это вполне реально, когда объект активизирован. После нажатия клавиши <Esc> деактивизированный объект будет отображаться в новой позиции.

Выборка объектов и работа с несколькими объектами

Следующим шагом будет программирование перехвата одиночных и двойных щелчков мышью, что позволит существенно упростить перемещение, изменение размеров и активизацию объекта. Для этого необходимо определить, где был выполнен щелчок — на помещенном в контейнер объекте или где-либо в другом месте экрана.

Анализ щелчков

Необходимо написать вспомогательную функцию, которая будет возвращать либо указатель на помещенный в контейнер объект, на котором пользователь выполнил щелчок мышью, либо значение NULL, если щелчок был выполнен на области окна представления, не содержащей помещенных в контейнер объектов. Эта функция будет работать со всеми объектами, помещенными в документ. Вставьте в файл ShowStringView.cpp сразу после определения деструктора текст программы, приведенный в листинге 14.26.

Листинг 14.26. Файл ShowStringView.cpp — метод CShowStringView::SetupTracker()

```
CShowStringCntrItem* CShowStringView::HitTest(CPoint point)
{
   CShowStringDoc* pDoc = GetDocument();
   CShowStringCntrItem* pHitItem = NULL;

   POSITION pos = pDoc->GetStartPosition();
    while (pos)
   {
      CShowStringCntrItem* pCurrentItem =
         (CShowStringCntrItem*) pDoc->GetNextClientItem(pos);
      if ( pCurrentItem->m_rect.PtInRect(point) )
      {
         pHitItem = pCurrentItem;
      }
   }
   return pHitItem;
}
```



Не забудьте добавить объявление этой открытой функции в файл заголовка.

Этой функции передается указатель на объект класса CPoint, определяющий то место на экране, где пользователь выполнил щелчок. Как мы уже видели ранее, каждый из помещенных в контейнер экземпляров объектов имеет собственный прямоугольник, описанный в m_rect, а класс CRect содержит метод PtInRect(), которому и передается объект CPoint. Если точка щелчка попадает в соответствующий прямоугольник, эта функция возвращает значение TRUE, в противном случае она возвращает значение FALSE. В приведенной программе организован цикл по всем помещенным в документ объектам, используя для этой цели метод класса документов OLE GetNextClientItem() и вызывая для каждого из них функцию PtInRect().

Что произойдет, если в документ помещено несколько экземпляров объектов, а пользователь сделал щелчок в таком месте экрана, где два или более объектов взаимно перекрываются? Выбран будет тот, который находится поверх всех. Это происходит потому, что функция GetStartPosition() возвращает указатель на самый нижний из элементов, а функция GetNextClientItem() обрабатывает объекты последовательно снизу вверх. Если в области, где пользователь выполнил щелчок мышью, два объекта перекрываются, указатель pHitItem будет помещен на тот из объектов, который находится внизу, а в следующей итерации цикла while он будет установлен на верхнем экземпляре объекта. Возвращен будет указатель на объект, который расположен сверху.

Отображение нескольких экземпляров объектов

Почему бы нам, анализируя фрагмент программы, в котором организован цикл по всем объектами, помещенным в контейнер, не откорректировать функцию CShowStringView::OnDraw() таким образом, чтобы она была способна выводить на экран все имеющиеся объекты? Оставьте неизменным текст программы, предназначенный для вывода на экран строки, и замените фрагмент функции, показанный в листинге 14.27, фрагментом, приведенным в листинге 14.28.

Листинг 14.27. Файл ShowStringView.cpp — подлежащие замене строки функции OnDraw()

```
// Вывод на экран осуществляется в произвольную позицию.
// Данный программный текст следует удалить, когда будет
// вставлен подготовленный вами реальный программный текст
// прорисовки. Указанная позиция в точности соответствует
// прямоугольнику, возвращаемому CShowStringCntrItem, что
// и создает эффект редактирования на месте.

// TODO: удалите этот программный текст, когда будет
// завершено создание реального текста прорисовки.
if (m_pSelection == NULL)
{
   POSITION pos = pDoc->GetStartPosition();
   m_pSelection = (CShowStringCntrItem*)pDoc->GetNextClientItem(pos);
}
if (m_pSelection != NULL)
{
   m_pSelection->Draw(pDC, m_pSelection->m_rect);
   CRectTracker trackrect;
   SetupTracker(m_pSelection,&trackrect);
   trackrect.Draw(pDC);
}
```

Листинг 14.28. Файл ShowStringView.cpp — строки, которые необходимо вставить в функцию nDraw()

```
POSITION pos = pDoc->GetStartPosition();
while (pos)
{
   CShowStringCntrItem* pCurrentItem =
      (CShowStringCntrItem*) pDoc->GetNextClientItem(pos);
   pCurrentItem->Draw(pDC, pCurrentItem->m_rect);

   if (pCurrentItem == m_pSelection )
   {
      CRectTracker trackrect;
      SetupTracker(pCurrentItem,&trackrect);
      trackrect.Draw(pDC);
   }
}
```

Теперь каждый из объектов будет выведен на экран, начиная с самого нижнего уровня и последовательно перемещаясь вверх. Если один из объектов выбран, он будет помещен в соответствующую прямоугольную рамку.

Обработка одиночных щелчков мышью

Когда пользователь делает щелчок кнопкой мыши в окне приложения, посылается сообщение WM_LBUTTONDOWN. Это сообщение должно быть перехвачено представлением. На вкладке ClassView сделайте щелчок правой кнопкой мыши на имени класса CShowStringView и выберите в появившемся контекстном меню команду Add Windows Message Handler. В расположенном слева поле New Windows Messages/Events щелкните на элементе WM_LBUTTONDOWN (рис. 14.10), а затем для немедленного редактирования текста программы щелкните на кнопке Add and Edit.

Добавьте текст программы, приведенный в листинге 14.29, в заготовку функции OnLButtonDown(), сгенерированную по команде Add Windows Message Handler.

Листинг 14.29. Файл ShowStringView.cpp — метод CShowStringView::OnLButtonDown()

```
void CShowStringView::OnLButtonDown(UINT nFlags, CPoint point)
{
   CShowStringCntrItem* pHitItem = HitTest(point);
   SetSelection(pHitItem);
   if (pHitItem == NULL)
      return;
   CRectTracker track;
   SetupTracker(pHitItem, &track);
   UpdateWindow();
   if (track.Track(this,point))
      {
         Invalidate();
         pHitItem->m_rect = track.m_rect;
         GetDocument()->SetModifiedFlag();
      }
}
```

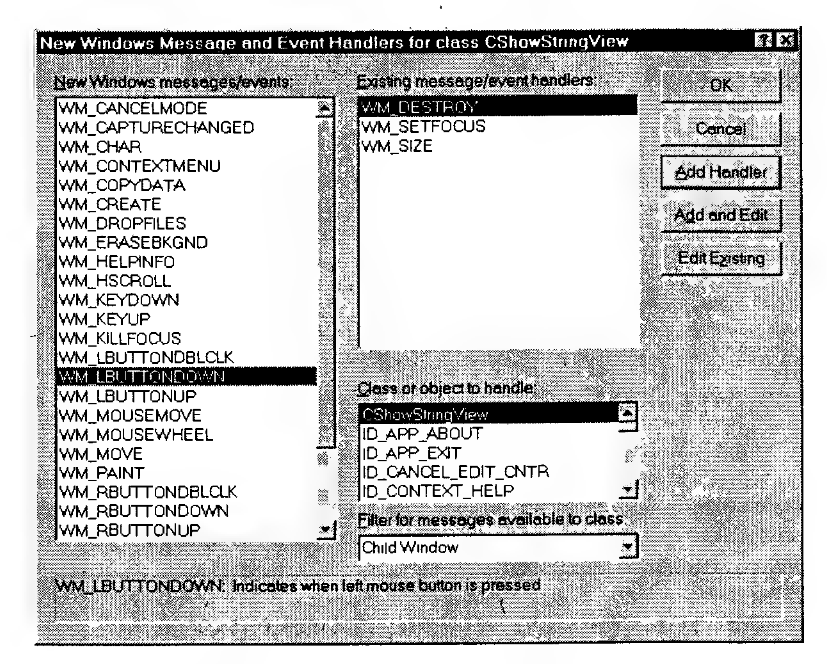


Рис. 14.10. Добавление функции для обработки щелчков левой кнопкой мыши

Этот фрагмент программы определяет, какой из объектов был выбран пользователем, а затем устанавливает для него состояние выбора. (Функция SetSelection() все еще не написана!) Далее, если что-то было выбрано, рисуется соответствующая рамка ндикации выборки и вызывается функция CRectTracker::Track(), обеспечивающая пользователю возможность изменять размеры объекта. Если пользователь изменяет размер рамки, объект соответственно переформатируется, после чего его изображение перерисовывается.

Алгоритм функции SetSelection() на удивление прямолинеен. Поместите прототип этой открытой функции-члена в файл заголовка ShowStringView.h и вставьте текст, приведенный в листинге 14.30, в файл ShowString.cpp.

Листинг 14.30. Файл ShowStringView.cpp — метод CShowStringView::SetSelection()

```
void CShowStringView::SetSelection(CShowStringCntrItem* item)
{
      // Если объект редактируется на месте, закройте его.
   if ( item == NULL || item != m_pSelection)
   {
      COleClientItem* pActive =
         GetDocument()->GetInPlaceActiveItem(this);
      if (pActive != NULL && pActive != item)
      {
         pActive->Close();
      }
   }
   Invalidate();
   m_pSelection = item;
}
```

Когда выбирается новый объект, любой редактирующийся на месте экземпляр объекта должен быть закрыт. Функция SetSelection() проверяет, не является ли объект, переданный ей при данном вызове, выбранным, и если не является, то находит объект, бывший в документе активным до текущего момента, после чего закрывает его. Затем вызывается перерисовка экрана и обновляется значение переменной m_pSelection.

Заново оттранслируйте приложение ShowString и запустите его на выполнение. Вставьте в документ объект и нажмите клавишу <Esc> для завершения редактирования на месте. Щелкните и перетащите куда-либо неактивный объект, а затем вставьте другой объект. У вас на экране должна получиться картинка, похожая на ту, которая приведена на рис. 14.11. Обратите внимание на размерные маркеры вокруг графического объекта, указывающие на то, что этот объект выбран.

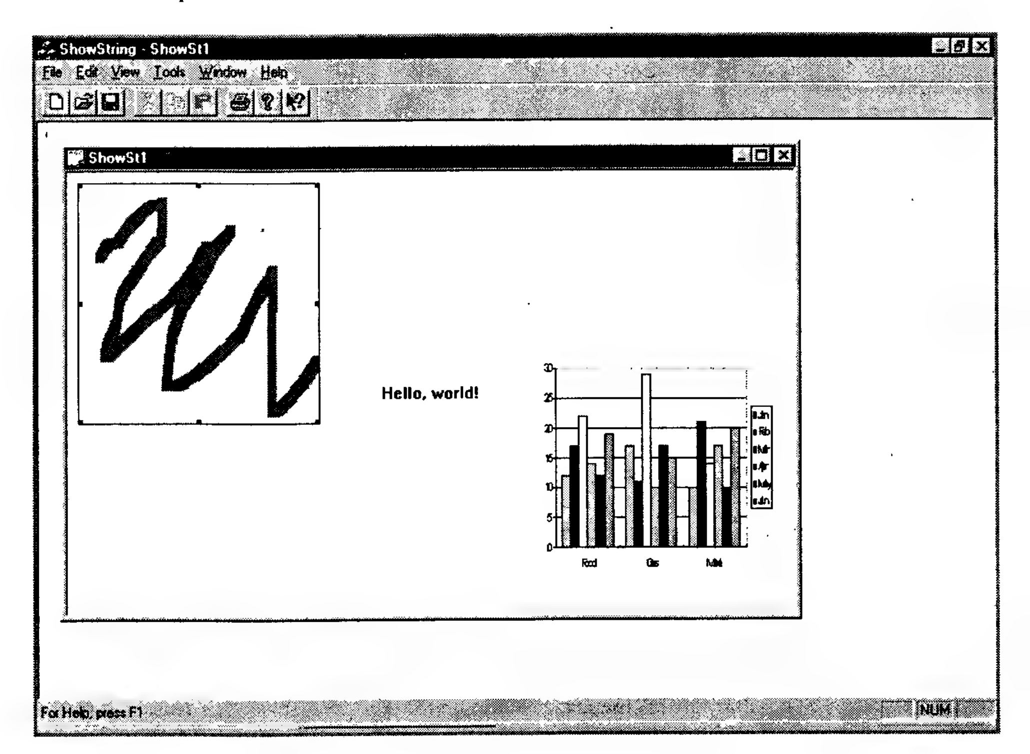


Рис. 14.11. Теперь приложение ShowString может поддерживать вставку нескольких объектов, а пользователю предоставлены возможности перемещения объектов и изменения их размеров с помощью мыши

Вы, вероятно, уже обратили внимание на то, что указатель мыши не изменяет своего вида при перемещении объекта или изменении его размеров. Это происходит по той простой причине, что вы еще не запрограммировали обратного. К счастью, это сделать несложно: класс CRectTracker включает метод SetCursor(), и все, что следует сделать, — вызвать его при появлении сообщения WM_SETCURSOR. И вновь перехват этого сообщения следует возложить на представление, поэтому на вкладке ClassView сделайте щелчок правой кнопкой мыши на классе CShowStringView, после чего в раскрывшемся контекстном меню выберите команду Add Windows Message Handler. В левой части диалогового окна в поле New Windows Messages/Events выберите значение WM_SETCURSOR, а затем, чтобы немедленно добавить новую функцию обработки сообщения и редактирования ее текста, щелкните на кнопке Add

and Edit. Поместите в сгенерированную заготовку функции строки, приведенные в листинге 14.31.

Листинг 14.31. Файл ShowStringView.cpp — метод CShowStringView::OnSetCursor()

```
BOOL CShowStringView::OnSetCursor(CWnd* pWnd, UINT nHitTest,
   UINT message)
{
   if (pWnd == this && m_pSelection != NULL)
   {
      CRectTracker track;
      SetupTracker(m_pSelection, &track);
      if (track.SetCursor(this, nHitTest))
      {
         return TRUE;
      }
   }
   return CView::OnSetCursor(pWnd, nHitTest, message);
}
```

В данном фрагменте программы ничего не выполняется, пока в представлении не потребуется изменить форму указателя мыши в связи с наличием выбранного объекта. В этом случае функции SetCursor() — члену объекта рамки выделения — предоставляется возможность изменять форму указателя мыши, поскольку именно объект рамки выделения знает, где на экране располагается его прямоугольник и находится ли указатель мыши на его границе или на размерном маркере. Если функция SetCursor() не изменит форму указателя мыши, данный программный фрагмент передаст управление указателем базовому классу. Заново оттранслируйте приложение ShowString, и вы увидите, что теперь при перемещении объекта и изменении его размеров изображение указателя мыши меняется, организуя, таким образом, обратную связь между программой и пользователем.

Обработка двойных щелчков мышью

Когда пользователь выполняет на помещенном в контейнер экземпляре объекта двойной щелчок мышью, вызывается *первичная команда* этого объекта. Для большинства объектов первичной командой является команда Edit in Place (Редактирование на месте), но для других, например звуковых файлов, это может быть команда Play (Запустить). Как и прежде, организуйте в классе CShowStringView перехват сообщения WM_LBUTTONDBLCLK и добавьте в новую заготовку функции OnLButtonDblClk() строки, приведенные в листинге 14.32.

Листинг 14.32. Файл ShowStringView.cpp — функция CShowStringView::OnLButtonDblClk()

```
void CShowStringView::OnLButtonDblClk(UINT nFlags, CPoint point)
{
   OnLButtonDown(nFlags, point);
   if( m_pSelection)
   {
      if (GetKeyState(VK_CONTROL) < 0)
      {
         m_pSelection->DoVerb(OLEIVERB_OPEN, this);
      }
      else
      {
         m_pSelection->DoVerb(OLEIVERB_PRIMARY, this);
```

```
        }
    }
    CView::OnLButtonDblClk(nFlags, point);
}
```

Сначала эта функция дублирует операции, которые запрограммированы при обработке обычного щелчка. Для этого вызывается функция OnLButtonDown(), которая выводит на экран рамку выделения, устанавливает значение переменной m_pSelection и т.д. Затем, если пользователь при выполнении двойного щелчка удерживал нажатой клавишу <Ctrl>, объект открывается для редактирования в отдельном окне. В противном же случае вызывается определенная для объекта первичная команда. И наконец, вызывается одноименная функция базового класса. Оттранслируйте и запустите приложение ShowString, попробуйте сделать двойной щелчок мышью. Вставьте объект, для прекращения его редактирования нажмите клавишу <Esc>, а затем сделайте на объекте двойной щелчок. Начнется процесс редактирования объекта на месте.

Реализация в приложении технологии "перетащить и опустить"

Последним шагом на пути превращения ShowString в современное полнофункциональное приложение-контейнер ActiveX будет реализация технологии "перетащить и опустить". Пользователь получит возможность с помощью мыши захватить объект, помещенный в контейнер, и переместить его, перетащив за пределы окна контейнера. Если при выполнении этой процедуры удерживать нажатой клавишу <Ctrl>, объект будет не перемещен, а скопирован с сохранением оригинала в контейнере. Пользователь получит возможность и обратного действия — перетащить объект из некоторого приложения в контейнер и поместить его в документ так, как это происходит при копировании с помощью буфера обмена. Другими словами, контейнер приобретет возможность функционировать как *источник перетаскивания* и как *адресат перетаскивания*.

Реализация функций источника перетаскивания

Поскольку класс CShowStringCntrItem наследуется от класса COleClientItem, реализация функций источника перетаскивания оказывается совсем простым делом. В файле ShowStringView.cpp отредактируйте текст функции CShowStringView::OnLButtonDown() так, как это показано в листинге 14.33. Новые строки выделены **полужирным** шрифтом.

Листинг 14.33. Метод CShowStringView::OnLButtonDown() — реализация функций источника перетаскивания

```
void CShowStringView::OnLButtonDown(UINT nFlags, CPoint point)
{
    CShowStringCntrItem* pHitItem = HitTest(point);
    SetSelection(pHitItem);
    if (pHitItem == NULL)
        return;

    CRectTracker track;
    SetupTracker(pHitItem, &track);
    UpdateWindow();

    if (track.HitTest(point) == CRectTracker::hitMiddle)
    {
        CRect rect = pHitItem->m_rect;
```

```
        CClientDC dc(this);
        OnPrepareDC(&dc);
        dc.LPtoDP(&rect); // Преобразование координат
            // логического прямоугольника в координаты
            // прямоугольника физического устройства.
        rect.NormalizeRect();
        CPoint newpoint = point - rect.TopLeft();

        DROPEFFECT dropEffect = pHitItem->DoDragDrop(rect, newpoint);
        if (dropEffect == DROPEFFECT_MOVE)
        {
            Invalidate();
            if (pHitItem == m_pSelection)
            {
                m_pSelection = NULL;
            }
            pHitItem->Delete();
        }
    }
    else
    {
        if (track.Track(this,point))
        {
            Invalidate();
            pHitItem->m_rect = track.m_rect;
            GetDocument()->SetModifiedFlag();
        }
    }
}
```

В добавленном фрагменте программы прежде всего проверяется, произошел ли щелчок мышью внутри рамки выделения, а не на размерных маркерах или на самой рамке. Далее создается временный объект класса CRect, который после выполнения некоторых преобразований координат будет передан функции DoDragDrop(). Первое преобразование состоит в переходе от логических координат прямоугольника к его физическим координатам и выполняется оно посредством вызова функции CDC::LPtoDP(). Чтобы обратиться к этой функции, в добавленном фрагменте программы на основе координат объекта класса CShowStringView, для которого вызывалась функция OnLButtonDown(), создается контекст временного прямоугольника. После преобразования логических координат объекта CRect в физические координаты на экране в добавленном фрагменте программы выполняется его нормализация и вычисление координат точки в прямоугольнике, на которой пользователь сделал щелчок.

Далее вызывается функция-член DoDropDrag() класса CShowStringCntrItem, унаследованная от класса COleClientItem() и не переопределенная в классе-потомке. Ей передается преобразованный объект CRect и координаты точки, на которой был выполнен щелчок. Если функция DoDragDrop() возвращает значение DROPEFFECT_MOVE, это означает, что объект был перемещен за пределы документа и его следует удалить. В фрагменте программы (который еще не написан), предназначенном для обработки сброса объекта после перетаскивания, будет создан новый экземпляр помещенного в контейнер объекта, который и будет определен как текущий выбранный элемент. Это означает, что, если исходный экземпляр объекта перетащить в иное место в том же приложении-контейнере, состояние текущей выборки будет снято с экземпляра объекта, на котором был сделан начальный щелчок. Если оба указателя на экземпляры объектов окажутся в конце операции идентичными, можно сделать вывод, что объект перетащили за пределы контейнера. Если объект был выведен за пределы контейнера, переменной m_pSelection во вновь добавленном программном тексте присваивается значение Null. Другими словами, объект, определенный указателем pHitItem, подлежит удалению.

Откомпилируйте приложение ShowString и запустите его на выполнение. Вставьте новый объект и нажмите клавишу <Esc> для выхода из режима редактирования объекта. Затем перетащите неактивный объект в какое-либо приложение, являющееся контейнером ActiveX, например Microsoft Excel. Можно попробовать перетащить объект на поверхность рабочего стола Windows 95. Проверьте работу следующего сценария: перетащите объект на панель задач и задержите его над пиктограммой свернутого контейнерного приложения. Когда это приложение будет восстановлено, у вас появится возможность опустить объект в его окне.

Реализация функций адресата перетаскивания

Реализовать в ShowString функцию адресата перетаскивания несколько сложнее (чего и следовало ожидать). Если вы в практическом примере перетащили помещенный в ShowString объект в какое-либо другое приложение, попробуйте перетащить его обратно в ShowString. Указатель мыши примет форму перечеркнутого круга, что означает *перетаскивать сюда объект запрещается*. В данном разделе мы выполним все необходимые изменения в тексте программы, чтобы подобная операция стала допустимой.

Во-первых, необходимо зарегистрировать наше представление как место, в которое можно выполнять перетаскивание экземпляров объектов. А во-вторых, необходимо обеспечить обработку следующих четырех событий, которые могут произойти при выполнении перетаскивания.

- Объект при перетаскивании пересекает границу окна вашего представления и попадает в его пределы. Это действие потребует изменения вида указателя мыши или какой-либо иной индикации, избранной вами по отношению к объекту.
- Объект перемещается в пределах окна представления. В этом случае необходимо обеспечить выдачу пользователю информации об этом процессе.
- Данный объект вновь покидает границы окна приложения, т.е. происходит транзитное прохождение окна данного приложения на пути к пункту назначения.
- Пользователь опускает объект в данном приложении.

Регистрация представления в качестве адресата перетаскивания

Для регистрации представления в качестве адресата перетаскивания добавьте в класс представления переменную-член типа COleDropTarget. Поместите приведенный ниже оператор объявления описания класса в файл ShowStringView.h:

```
COleDropTarget m_droptarget;
```

Для обработки регистрации переопределите в представлении функцию OnCreate(), которая вызывается при его создании. Организуйте перехват сообщения WM_CREATE в классе CShowStringView. В заготовку функции, сгенерированную для обработки этого сообщения, поместите текст программы, приведенный в листинге 14.34.

Листинг 14.34. Файл ShowStringView.cpp — функция CShowStringView::OnCreate()

```
int CShowStringView::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
   if (CView::OnCreate(lpCreateStruct) == -1)
      return -1;
```

```
    if (m_droptarget.Register(this))
        {
            return 0;
        }
    else
        {
            return -1;
        }
}
```

Функция OnCreate() возвращает значение 0, если все происходит нормально, и значение –1, если окно должно быть закрыто. Данная подпрограмма вызывает функцию базового класса, а затем использует функцию COleDropTarget::Register() для регистрации данного представления в качестве места, куда можно перетаскивать объекты.

Определение основных функций и добавление переменных-членов

Четырем событиям, которые могут происходить в приложении, соответствуют четыре виртуальные функции, которые требуется переопределить: OnDragEnter(), OnDragOver(), OnDragLeave() и OnDrop(). Чтобы выполнить их переопределение, в окне ClassView сделайте щелчок правой кнопкой мыши на имени класса CShowStringView, а затем выберите Add Virtual Function. В списке New Virtual Function выделите функцию OnDragEnter() и щелкните на кнопке Add Handler. Повторите эти действия для оставшихся трех функций.

При перетаскивании функция OnDragEnter() создает *фокусную рамку* (focus rectangle), указывающую на то место, где будет вставлен данный объект, если пользователь отпустит кнопку мыши. Вывод и перемещение фокусной рамки по экрану производится функцией OnDragOver(). Однако прежде всего следует определить в классе CShowStringView несколько переменных-членов, имеющих отношение к фокусной рамке. Вставьте приведенные ниже строки в файл ShowStringView.h в раздел public:

```
CPoint m_dragpoint;
CSize m_dragsize;
CSize m_dragoffset;
```

Объект данных содержит большой объем информации о самом себе, представленной в различных форматах. Сюда входят, конечно, собственно данные, такие как текст, *независимое от устройства растровое изображение* (DIB — Device Independent Bitmap) или что-либо иное в соответствующем формате. Но имеется и информация о самом объекте. Если вы затребуете данные в *формате описания объекта* (Object Descriptor Format), то сможете получить сведения о размере того объекта, на изображении которого пользователь выполнил начальный щелчок, и расстояние от указателя мыши до верхнего левого угла объекта. Вся эта информация обычно именуется данными в *формате буфера обмена* (Clipboard Format), поскольку она используется для вырезания и вставки с помощью буфера обмена.

Для запроса этой информации вызывается функция GetGlobalData() — член объекта, которой передается параметр, означающий требование передачи дескриптора объекта. Вместо того чтобы формировать этот параметр из текстовой строки всякий раз при обращении к функции, он строится один раз и сохраняется в статической переменной класса. Когда для класса определена статическая переменная-член, каждый экземпляр данного класса для получения ее значения обращается к одной и той же области памяти. Она инициализируется (и для нее выделяется память) один раз, вне пределов класса.

Добавьте следующую строку в файл ShowStringView.h:

```
static CLIPFORMAT m_cfObjectDescriptorFormat;
```

В файл ShowStringView.cpp сразу после текста первой функции поместите следующий оператор:

```
CLIPFORMAT CShowStringView::m_cfObjectDescriptorFormat =
   (CLIPFORMAT) ::RegisterClipboardFormat("Object Descriptor");
```

Этот оператор преобразует строку "Object Descriptor" в тип CLIPFORMAT и сохраняет ее в статической переменной-члене для использования любым экземпляром данного класса. Работа со статической переменной в представлении позволяет ускорить процесс перетаскивания.

Возможно, ваше представление не должно принимать сколько угодно экземпляров объектов произвольнюго типа, которые пользователь может попытаться в него перетащить. Добавьте в класс представления логическую переменную-член, предназначенную для указания, может ли быть принят тот объект, который пользователь в данный момент перетаскивает через окно представления:

```
BOOL m_OKtodrop;
```

Осталась одна последняя переменная-член, которую необходимо добавить в класс представления CShoiwStringView. Во время перетаскивания пользователем объекта через окно представления фокусная рамка постоянно рисуется и стирается. Добавьте еще одну логическую переменную-член для отслеживания состояния фокусной рамки:

```
BOOL m_FocusRectangleDrawn;
```

В конструкторе представления инициализируйте переменную m_FocusRectangleDrawn значением FALSE:

```
CShowStringView::CShowStringView()
{
   m_pSelection = NULL;
   m_FocusRectangleDrawn = FALSE;
}
```

Метод OnDragEnter()

Метод OnDragEnter() вызывается, когда пользователь при перетаскивании объекта первый раз пересекает границу окна представления. Этот метод создает фокусную рамку, а затем вызывает функцию OnDradOver(). Функция OnDragOver() будет постоянно вызываться при продолжении движения объекта, пока пользователь не выведет его за пределы окна представления или не опустит его над ним. Общая структура метода OnDragEnter() показана в листинге 14.35.

Листинг 14.35. Файл ShowStringView.cpp — метод CShowStringView::OnDragEnter()

```
DROPEFFECT CShowStringView::OnDragEnter(COleDataObject* pDataObject,
    DWORD dwKeyState, CPoint point)
{
   ASSERT(!m_FocusRectangleDrawn);

      // Выполняется проверка допустимости перетаскивания объекта
   // над окном данного представления.


   // С помощью вызова функции GetGlobalData устанавливаются
   // значения dragsize и dragoffset.

   // Преобразование относительных координат в пиксели.
```

```
   // Управление передается функции OnDragOver.

   return OnDragOver(pDataObject, dwKeyState, point);
}
```

Прежде всего проверяется, можно ли из того объекта, на который ссылается указатель pDataObject, создать экземпляр класса COleClientItem (и, следовательно, экземпляр класса CShowStringCntrItem). Если это невозможно, объект не может быть вставлен в данный документ и функция возвращает значение DROPEFFECT_NONE, как показано в листинге 14.36.

Листинг 14.36. Файл ShowStringView.cpp — фрагмент OnDragEnter(), выполняющий проверку на допустимость вставки объекта

```
// Выполняется проверка допустимости перетаскивания объекта
// над окном данного представления.

m_OKtodrop = FALSE;
if (!COleClientItem::CanCreateFromData(pDataObject))
   return DROPEFFECT_NONE;
m_OKtodrop = TRUE;
```

Наступило время выполнить некоторые очень важные манипуляции. Для получения информации, описывающей объект, вызывается функция-член GetGlobalData() объекта, пересекающего при перетаскивании границу данного представления. Эта функция возвращает дескриптор блока глобальной памяти. Затем для преобразования дескриптора в указатель на первый байт блока и введения запрета на распределение этого блока для любого другого объекта вызывается функция SDK GlobalLock(). В результате мы получили указатель на структуру, содержащую описание объекта (любопытствующие могут выяснить состав данной структуры, ознакомившись приблизительно с двумя тысячами строк ее описания в файле oleidl.h, обычно хранящемся в папке C:\MSDEV\include). Используем элементы sizel и pointl этой структуры для установки значений переменных-членов m_dragsize и m_dragoffset.



В именах элементов структуры последний знак является не цифрой 1, а строчной буквой l. Элементами структуры sizel являются переменные cx и cy, а элементами структуры pointl являются переменные x и y. Не потеряйте и не перепутайте эти имена в процессе вырезания и вставки текста.

И наконец, функция GlobalUnlock() отменяет все установки, выполненные функцией GlobalLock(), разрешая другим объектам доступ к блоку памяти, а функция GlobalFree() освобождает память. Текст всей процедуры в целом выглядит так, как показано в листинге 14.37.

Листинг 14.37. Файл ShowStringView.cpp — фрагмент OnDragEnter(), выполняющий установку переменных dragsize и dragoffset

```
// С помощью вызова функции GetGlobalData устанавливаются
// значения dragsize и dragoffset.

HGLOBAL hObjectDescriptor = pDataObject->GetGlobalData(
   m_cfObjectDescriptorFormat);
if (hObjectDescriptor)
{
   LPOBJECTDESCRIPTOR pObjectDescriptor =
      (LPOBJECTDESCRIPTOR) GlobalLock(hObjectDescriptor);
```

```
    ASSERT(pObjectDescriptor);
    m_dragsize.cx = (int) pObjectDescriptor->sizel.cx;
    m_dragsize.cy = (int) pObjectDescriptor->sizel.cy;
    m_dragoffset.cx = (int) pObjectDescriptor->pointl.x;
    m_dragoffset.cy = (int) pObjectDescriptor->pointl.y;
    GlobalUnlock(hObjectDescriptor);
    GlobalFree(hObjectDescriptor);
}
else
{
    m_dragsize = CSize(0,0);
    m_dragoffset = CSize(0,0);
}
```

На заметку

Глобальная память, иначе называемая *разделяемой памятью приложения*, распределяется в месте, отличном от того, где располагается память, доступная для обычной обработки. Эта память используется при возникновении необходимости выполнения записи и чтения одной и той же области памяти двумя различными процессами. Данный тип памяти широко применяется в приложениях ActiveX.

Для некоторых операций ActiveX размер глобальной памяти оказывается слишком мал — представьте себе попытку передать через глобальную память файл размером 40 Мбайт! Существует более общая по сравнению с GetGlobalData() функция, называемая (и это не удивительно) GetData(), которая может передавать данные при помощи самых разнообразных носителей. Поскольку описания объектов невелики по размеру, передача их через глобальную память является вполне разумным подходом.

Если попытка получить память путем вызова функции GetGlobalData() оказывается неудачной, обеим переменным-членам с помощью нулевых прямоугольников присваиваются нулевые значения. Следующий шаг — преобразование полученных прямоугольников из координат OLE (которые независимы от устройства) в пиксели:

```
// Преобразование относительных координат в пиксели.

    CClientDC dc(NULL);
    dc.HIMETRICtoDP(&m_dragsize);
    dc.HIMETRICtoDP(&m_dragoffset);
```

Функция HIMETRICtoCP() — это очень полезная функция, которая, к счастью, является членом класса CClientDC, наследующего от класса CDC, знакомого нам по главе 6, *Вывод на экран*. Как только создан экземпляр класса CClientDC, данная функция становится доступной.

Функция OnDragEnter() завершает работу вызовом функции OnDragOver(), которую мы сейчас рассмотрим.

Функция OnDragOver()

Данная функция возвращает значение типа DROPEFFECT. Как уже отмечалось выше, если возвращается значение DROPEFFECT_MOVE, источник перетаскивания удаляет из своего документа данный объект. Возвращение значения DROPEFFECT_NONE означает отмену копирования. Именно функция OnDrawOver() осуществляет подготовку к приему объекта или отказу от его копирования. Общая структура функции выглядит следующим образом.

```
DROPEFFECT CShowStringView::OnDragOver(COleDataObject* pDataObject,
    DWORD dwKeyState, CPoint point)
{
    // Выход, если сброс объекта уже отклонен.

        // Определение по нажатыми клавишами типа включения объекта.
```

```
    // Выставление фокусной рамки.
}
```

Прежде всего следует проверить, не отклонено ли включение данного объекта функцией OnDragEnter() или же при предыдущем вызове OnDragOver():

```
// Выход, если сброс объекта уже отклонен.
if (!m_OKtodrop)
{
    return DROPEFFECT_NONE;
}
```

Затем анализируется, не удерживает ли пользователь какие-либо клавиши нажатыми в данный момент. Информацию о клавишах можно получить из аргумента dwKeyState. Предлагаемое решение поставленной задачи (листинг 14.38) вполне реально.

Листинг 14.38. Файл ShowStringView.cpp — фрагмент OnDragOver(), выполняющий определение типа вставки объекта

```
// Определение по нажатыми клавишами типа включения объекта.
DROPEFFECT dropeffect = DROPEFFECT_NONE;

if ((dwKeyState & (MK_CONTROL|MK_SHIFT) )
    == (MK_CONTROL|MK_SHIFT))
{
    // Ctrl+Shift - требование выполнить связывание.
    dropeffect = DROPEFFECT_LINK;
}

else if ((dwKeyState & MK_CONTROL) == MK_CONTROL)
{
    // Ctrl - требование выполнить копирование.
    dropeffect = DROPEFFECT_COPY;
}
else if ((dwKeyState & MK_ALT) == MK_ALT)
{
    // Alt - требование выполнить пересылку.
    dropeffect = DROPEFFECT_MOVE;
}
else
{
    // По умолчанию выполняется пересылка.
    dropeffect = DROPEFFECT_MOVE;
}
```

На заметку

Этот фрагмент программы должен быть намного сложнее, если размер документа меньше размера представления. Подобное может происходить, когда вы редактируете растровую картинку в Paint; особенно часто это происходит, когда представление можно прокручивать. В примере приложения-контейнера ActiveX DRAWCLI, приводимом Microsoft (он включен в компакт-диск Visual C++), эти ситуации обрабатываются. Найдите файл drawvw.cpp в папке \DevStudio\Vc\Samples\Mci\Ole\ DrawCli и сравните данный фрагмент с соответствующим фрагментом его функции OnDragOver().

Если объект со времени последнего вызова OnDragOver() был перемещен, фокусную рамку следует удалить, а затем вновь нарисовать в новой позиции. Поскольку фокусная рамка выводится простой операцией XOR над отдельными цветами, нарисовать ее второй раз на том же месте — это все равно, что удалить ее. Фрагмент функции, управляющий выводом фокусной рамки, приведен в листинге 14.39.

Листинг 14.39. Файл ShowStringView.cpp — фрагмент OnDragOver(), осуществляющий управление фокусной рамкой

```
// Управление отображением фокусной рамки.

point -= m_dragoffset;
if (point == m_dragpoint)
{
   return dropeffect;
}

CClientDC dc(this);

if (m_FocusRectangleDrawn)
{
   dc.DrawFocusRect(CRect(m_dragpoint, m_dragsize));
   m_FocusRectangleDrawn = FALSE;
}

if (dropeffect != DROPEFFECT_NONE)
{
dc.DrawFocusRect(CRect(point, m_dragsize));
m_dragpoint = point;
m_FocusRectangleDrawn = TRUE;
}
```

Чтобы проверить, следует ли проводить перерисовку фокусной рамки, в данном фрагменте текста используются координаты точки, на которой сделал щелчок пользователь. На основании этих координат вычисляется положение верхнего левого угла объекта, которое сравнивается с позицией верхнего левого угла фокусной рамки. Если положения обеих точек идентичны, перерисовка рамки не требуется. Если координаты верхних левых углов различны, фокусную рамку следует удалить с экрана.

На заметку При первом вызове функции OnDragOver() переменная m_dragpoint не определена. Но это не важно, поскольку переменная m_FocusRectangleDrawn имеет значение FALSE и макрос ASSERT в функции OnDragEnter() нам это гарантирует. Когда переменная m_FocusRectangleDrawn получит значение TRUE, одновременно с ней значение будет помещено и в переменную m_dragpoint.

Последний штрих — замените автоматически сгенерированную команду выхода такой командой, которая будет возвращать вычисленное значение типа DROPEFFECT:

```
return dropeffect;
```

Метод OnDragLeave()

В некоторых случаях пользователь может транзитом перетащить объект над окном приложения и выйти за его пределы. Метод OnDragLeave() должен всего лишь удалить фокусную рамку, что и продемонстрировано в листинге 14.40.

Листинг 14.40. Файл ShowStringView.cpp — метод CShowStringView::OnDragLeave()

```
void CShowStringView::OnDragLeave()
{
   CClientDC dc(this);
   if (m_FocusRectangleDrawn)
   {
      dc.DrawFocusRect(CRect(m_dragpoint, m_dragsize));
```

```
        m_FocusRectangleDrawn = FALSE;
    }
}
```

Метод OnDragDrop()

Если пользователь отпускает кнопку мыши и опускает перетаскиваемый объект над окном ShowString, объект помещается в контейнер и вызывается метод OnDragDrop(). Общая структура этого метода приведена в листинге 14.41.

Листинг 14.41. Файл ShowStringView.cpp — структура CShowStringView::OnDragDrop()

```
BOOL CShowStringView::OnDrop(COleDataObject* pDataObject,
    DROPEFFECT dropEffect, CPoint point)
{
    ASSERT_VALID(this);
    // Удаление фокусной рамки.

    // Вставка объекта данных.

    // Определение размеров объекта и установка его как текущей выборки.

    // Обновление представления и установка флажка модификации.

    return TRUE;
}
```

Удаление фокусной рамки является делом несложным; соответствующий фрагмент программы показан в листинге 14.42.

Листинг 14.42. Файл ShowStringView.cpp — фрагмент OnDragDrop(), выполняющий удаление фокусной рамки

```
// Удаление фокусной рамки.

CClientDC dc(this);
if (m_FocusRectangleDrawn)
{
    dc.DrawFocusRect(CRect(m_dragpoint, m_dragsize));
    m_FocusRectangleDrawn = FALSE;
}
```

Затем для хранения данных вставляемого объекта создается новый экземпляр класса объекта, как показано в листинге 14.43. Обратите внимание на использование битовых масок и операции побитового И (&) для проверки наличия связи.

Листинг 14.43. Файл ShowStringView.cpp — фрагмент OnDragDrop(), выполняющий вставку объекта в документ

```
// Вставка объекта данных.

CShowStringDoc* pDoc = GetDocument();
CSnowStringCntrItem* pNewItem = new CShowStringCntrItem(pDoc);
ASSERT_VALID(pNewItem);
if (dropEffect & DROPEFFECT_LINK)
{
    pNewItem->CreateLinkFromData(pDataObject);
```

```
}
else
{
    pNewItem->CreateFromData(pDataObject);
}
ASSERT_VALID(pNewItem);
```

Размер помещенного в контейнер объекта определяется так, как показано в листинге 14.44.

Листинг 14.44. Файл ShowStringView.cpp — фрагмент OnDragDrop(), определяющий размеры объекта

```
// Определение размеров объекта и установка его как текущей выборки.
CSize size;
pNewItem->GetExtent(&size, pNewItem->GetDrawAspect());
dc.HIMETRICtoDP(&size);
point -= m_dragoffset;
pNewItem->m_rect = CRect(point,size);
m_pSelection = pNewItem;
```

Обратите внимание на то, что данный фрагмент кода корректирует место, куда пользователь сбрасывает объект, соответственно содержимому переменной m_dragoffset. Последняя содержит координаты точки объекта, на которой пользователь впервые выполнил щелчок.

И последнее — необходимо указать, что документ при выходе из программы должен быть сохранен, поскольку вставка в контейнер нового объекта изменила его данные. Кроме того, следует заново вывести на экран представление.

```
// Обновление представления и установка флажка модификации.

pDoc->SetModifiedFlag();
pDoc->UpdateAllViews(NULL);
return TRUE;
```

Данная функция всегда возвращает значение TRUE, так как в момент выхода отсутствует контроль ошибок, который мог бы вызвать возврат значения FALSE. Обратите внимание на то, что мы заранее устранили возможность появления большинства проблем. Например, если объект нельзя помещать в контейнер, то функция OnDragEnter() возвращает значение DROPEFFECT_NONE и соответствующий фрагмент программы вообще не выполняется. Можете быть уверены в том, что приведенная программа вполне работоспособна.

Проверка функционирования приложения в качестве адресата перетаскивания

Доверяй, но проверяй. Любая уверенность априори не сможет заменить тестирования. Оттранслируйте и запустите на выполнение приложение ShowString, а затем попытайтесь перетащить в него что-нибудь. Для одновременного контроля над функционированием приложения и как источника, и как адресата перетаскивания перетащите что-либо сначала из приложения за пределы его окна, а потом верните это обратно. Шаг за шагом наше приложение превращается в очень полезный контейнер. Осталось решить последнюю задачу.

Удаление объекта

Удалить объект из нашего приложения можно, перетащив его за пределы окна представления. Однако необходимо реализовать и команду удаления объектов как способ более привычный. Командой, которая в общем случае используется для подобных целей, является Edit⇨Delete. Поэтому начнем со вставки этой команды в меню IDR_SHOWSTTYPE, поместив ее перед командой Insert New Object. Измените имя ID_EDIT_DELETE, предлагаемое Visual Studio в качестве идентификатора команды, на ID_EDIT_CLEAR, которое традиционно присваивается как идентификатор ресурса командам, удаляющим объект, помещенный в контейнер. Перейдите к другой команде меню, а затем вновь вернитесь к Edit⇨Delete, и вы увидите, что в строку сообщения было автоматически внесено значение Erase tne selection (Стирание выборки).

Обработка данной команды должна выполняться в классе представления, поэтому добавьте в него соответствующий обработчик сообщения, как мы уже неоднократно делали это ранее в данной главе. Выполните следующие действия.

1. Раскройте окно ClassWizard.
2. Выберите значение ID_EDIT_CLEAR в списке Class or Object to Handle, расположенном справа в нижней части окна.
3. В списке New Windows Messages/Events выберите значение Command, которое должно появиться после щелчка на ID_EDIT_CLEAR.
4. Щелкните на кнопке Add Handler.
5. Для принятия предложенного имени щелкните на кнопке OK.
6. В списке New Windows Messages/Events выберите значение UPDATE_COMMAND_UI и вновь щелкните на кнопке Add Handler.
7. Примите имя, предложенное вам по умолчанию.
8. Для завершения процедуры щелкните на кнопке OK большого диалогового окна.

Тексты программ этих двух обработчиков сообщений чрезвычайно просты. Поскольку обработчик обновления CSnowStringView::OnUpdateEditClear() проще, начнем с него:

```
void CSnowStringView::OnUpdateEditClear(CCmdUI* pCmdUI)
{
   pCmdUI->Enable(m_pSelection != NULL);
}
```

Если в окне приложения присутствует выделенный объект, его можно будет удалить. Если в окне на данный момент ничего не выделено, команда удаления в меню станет неактивной (этот пункт будет затенен). Программа обработки этой команды не намного длиннее предыдущей (листинг 14.45).

Листинг 14.45. Файл ShowStringView.cpp — метод CShowStringView::OnEditClear()

```
void CSnowStringView::OnEditClear()
{
   if (m_pSelection)
   {
      m_pSelection->Delete();
      m_pSelection = NULL;
      GetDocument()->SetModifiedFlag();
      GetDocument()->UpdateAllViews(NULL);
   }
}
```

В данной функции проверяется, есть ли выборка на данный момент (даже если команда меню затенена и выборка в окне отсутствует). Затем выбранный объект удаляется; указателю выборки присваивается NULL, сигнализируя, что ничего на экране не выбрано; документ помечается как модифицированный, так что пользователю при выходе из приложения будет предложено сохранить документ. А затем экран перерисовывается уже без удаленного объекта.

Откомпилируйте и запустите на выполнение приложение SnowString. Вставьте что-нибудь в документ, а затем удалите этот объект, выбрав команду Edit⇨Delete. Мы закончили создание полнофункционального приложения-контейнера, оснащенного современным интуитивно понятным интерфейсом и способного выполнить все, что положено уметь контейнеру ActiveX.

ГЛАВА 15

Создание приложения-сервера ActiveX

В этой главе...

Добавление в ShowString функций сервера ActiveX

Приложения контейнер/сервер

Документы ActiveX

Мастер AppWizard может создавать как приложения-контейнеры ActiveX, так и приложения-серверы ActiveX. Однако в отличие от контейнеров текст программы, создаваемый AppWizard для серверов, является практически законченным и потребуется совсем немного времени для его окончательной шлифовки. В этой главе создается новая версия ShowString, являющаяся только сервером ActiveX, а также обсуждается, как создать еще одну версию, которая будет одновременно и сервером, и контейнером. Кроме того, мы рассмотрим, что такое документ ActiveX и как он используется в других приложениях.

Добавление в ShowString функций сервера ActiveX

Как и предыдущая, данная глава начинается с создания с помощью AppWizard простейшего приложения-сервера ActiveX с последующим добавлением к нему функциональных возможностей, превращающих это приложение в ShowString. Поскольку размеры самого приложения ShowString невелики, это будет намного быстрее, чем добавлять функциональные возможности сервера ActiveX к обычному приложению ShowString.

Исходный вариант сервера ActiveX, формируемый AppWizard

Создайте новое приложение ShowString, разместив его в отдельном каталоге и выполнив в процессе настройки мастера практически те же установки параметров, которые были сделаны при создании прежних версий ShowString в главах 8 и 14. Присвойте создаваемому приложению имя ShowString и выберите вариант приложения класса MDI без поддержки операций с базами данных. На третьем этапе настройки AppWizard закажите создание полнофункционального сервера ActiveX как варианта поддержки составных документов. Такой выбор снимает блокировку флажка ActiveX document server (сервер документов ActiveX), но его пока что устанавливать не следует. Условия, при которых следует устанавливать эту опцию, мы рассмотрим позднее в этой же главе. Продолжайте работу с AppWizard, выбрав опции стационарной панели инструментов, строки состояния, печати и предварительного просмотра распечатки, контекстно-зависимой справки и трехмерного дизайна элементов управления. На последнем этапе установите создание комментариев и использование разделяемой библиотеки DLL.

На заметку Хотя в настоящее время обсуждаемая технология называется ActiveX, на страницах мастера AppWizard используются ссылки на технологию поддержки составных документов. Кроме того, в именах многих классов, которые используются в этой главе, содержится аббревиатура OLE. Несмотря на то что Microsoft уже изменила название этой технологии, соответствующие изменения в Visual C++ еще не проведены. Придется смириться с указанными противоречиями вплоть до выхода следующей версии Visual C++.

Существует множество различий между приложением, которое вы только что создали, и заготовкой приложения, не обеспечивающего поддержки функций сервера ActiveX. Данные различия поясняются в нескольких последующих подразделах.

Меню

В приложении-сервере ActiveX появились два новых меню. Первое, с идентификатором IDR_SHOWSTTYPE_SRVR_IP, показано на рис. 15.1. Когда объект редактируется на месте, меню редактирования на месте контейнера (это меню в контейнерной версии ShowString имеет

идентификатор IDR_SHOWSTTYPE_CNTR_IP) объединяется с меню редактирования на месте сервера IDR_SHOWSTTYPE_SRVR_IP, и в результате создается *объединенное* меню редактирования на месте, показанное на рис. 15.2. Две вертикальные линии в каждом меню являются точками объединения.

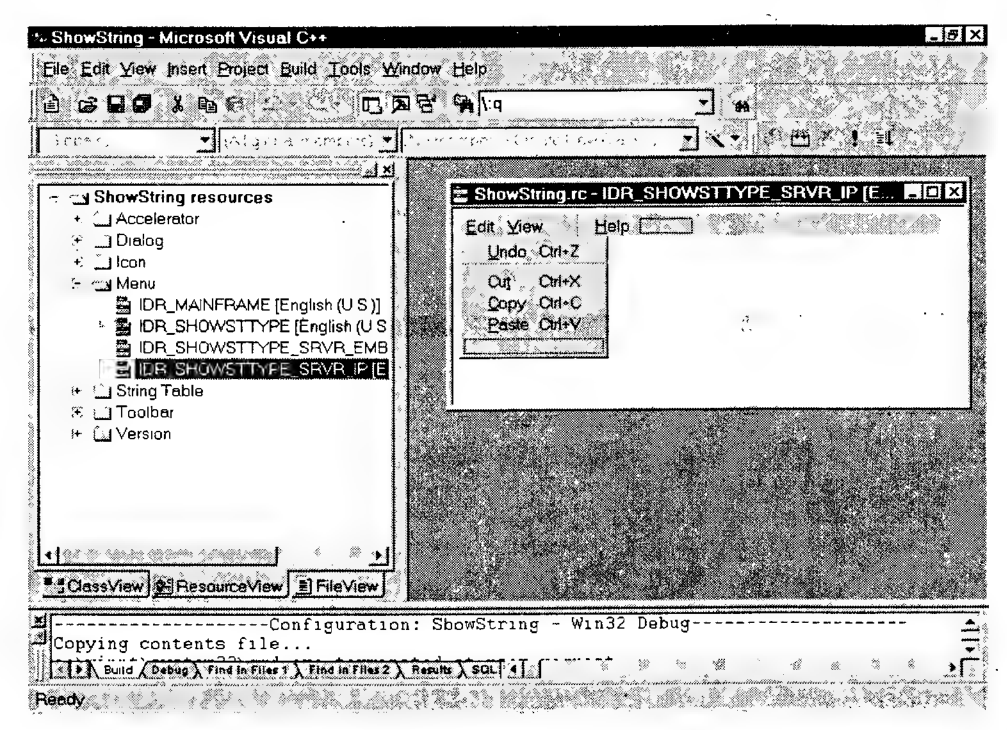


Рис. 15.1. Мастер AppWizard добавляет в приложение новое меню, предназначенное для редактирования объекта на месте

Второе новое меню с идентификатором IDR_SHOWSTTYPE_SRVR_EMB используется, когда внедренный объект редактируется в отдельном окне. Это меню показано на рис. 15.3, где оно расположено рядом с уже знакомым нам меню IDR_SHOWSTTYPE. Последнее использовалось в ShowString, когда оно было обычным приложением, а не сервером ActiveX. Меню File в этих двух случаях содержит разные команды: меню IDR_SHOWSTTYPE_SVR_EMB содержит команду Update вместо команды Save и команду Save Copy As вместо команды Save As. Это происходит по той причине, что экземпляр объекта, с которым пользователь работает в отдельном окне, является не самостоятельным документом, а лишь объектом, внедренным в другой документ. Выбор команды File⇨Update приведет к обновлению внедренного объекта, а выбор команды File⇨Save Copy As вызовет не сохранение всего документа, а лишь копирование внедренного объекта в отдельный файл.

Класс CShowStringApp

В этот класс добавлена новая переменная-член. Она объявляется в файле showstring.h следующим образом:

```
COleTemplateServer m_server;
```

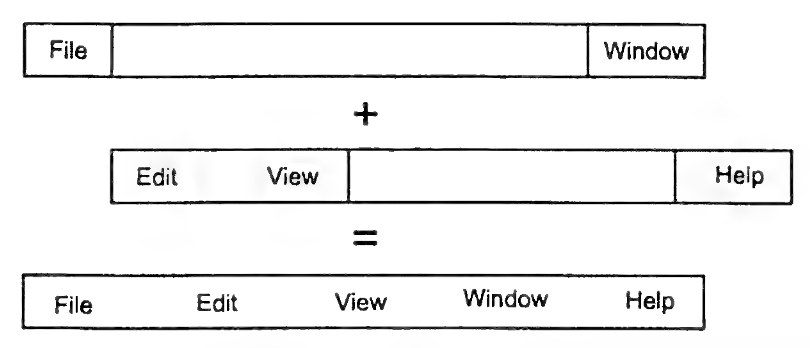


Рис. 15.2. При редактировании объекта на месте меню контейнера и сервера объединяются в единое меню

Рис 15.3 В меню File панели меню, выводимой в окно при редактировании внедренных объектов, содержатся команды, отличные от тех, которые присутствуют в меню File обычных окон

Класс `COleTemplateServer`, как мы увидим ниже, выполняет большую часть работы, связанной с подключением документа к серверу для обработки.

В начало файла `showstring.cpp` добавлена директива:

```
#include "IpFrame.h"
```

Она подключает к программе определение класса `CInPlaceFrame`, обсуждаемого позднее в этой главе. Сразу после описания функции `InitInstance()` добавлены строки, представленные в листинге 15.1.

Листинг 15.1. Фрагмент файла ShowString.cpp — определение CLSID приложения

```
// Этот идентификатор был специально сгенерирован для вашего
// приложения с целью обеспечения его статистической
// уникальности. Можете его изменить, если предпочитаете
// присвоить приложению конкретное значение идентификатора.

// {0B1DEE40-C373-11CF-870C-00201801DDD6}
static const CLSID clsid =
{ 0xb1dee40, 0xc373, 0x11cf,
   { 0x87, 0xc, 0x0, 0x20, 0x18, 0x1, 0xdd, 0xd6 } };
```

В программе, сгенерированной на вашем компьютере, эти числа будут иметь другие значения. Данный идентификатор класса (Class ID — CLSID) уникально определяет ваше серверное приложение и связанный с ним тип документов. Приложения, поддерживающие обработку нескольких типов документов (например, текст и графику), используют различные CLSID для каждого из типов документов.

В отличие от не поддерживающего ActiveX приложения ShowString, разработанного в главе 8, в функцию CShowStringApp::InitInstance() внесено несколько изменений (как и в случае контейнерной версии ShowString). Операторы, представленные в листинге 15.2, предназначены для инициализации библиотек ActiveX (OLE).

Листинг 15.2. Фрагмент файла ShowString.cpp — инициализация библиотек ActiveX

```
// Инициализация библиотек ActiveX (OLE).
if (!AfxOleInit())
{
   AfxMessageBox(IDP_OLE_INIT_FAILED);
   return FALSE;
}
```

В той же функции CShowStringApp::InitInstance() после инициализации MultiDocTemplate, но до вызова функции AddDocTemplate() добавлены следующие строки, которые предназначены для регистрации меню, используемых при редактировании на месте или при редактировании в отдельном окне.

```
pDocTemplate->SetServerInfo(
   IDR_SHOWSTTYPE_SRVR_EMB, IDR_SHOWSTTYPE_SRVR_IP,
   RUNTIME_CLASS(CInPlaceFrame));
```

Изменение, которое отсутствовало в контейнерной версии, состоит в подключении к шаблону документа идентификатора класса.

```
// Подключение класса COleTemplateServer к шаблону документа.
// Класс COleTemplateSever создает по запросам контейнеров OLE
// новые документы на основе информации, определенной в шаблоне
// документа.

m_server.ConnectTemplate(clsid, pDocTemplate, FALSE);
```

Теперь, когда серверное приложение будет запускаться для редактирования экземпляра объекта на месте или в отдельном окне, системная DLL добавит в командную строку вызова приложения параметр /Embedded. Но если приложение уже работает и является приложением с многодокументным интерфейсом, новая копия приложения запущена не будет. Вместо этого в данном приложении откроется новое окно. Все эти довольно-таки магические действия производятся вызовом всего одной функции, как показано в листинге 15.3.

Листинг 15.3. Фрагмент файла ShowString.cpp — регистрация выполнения MDI-приложения

```
// Регистрация сервера OLE как работающего.
// Это дает возможность библиотекам OLE создавать объекты,
// поддерживаемые другими приложениями.
COleTemplateServer::RegisterAll();

// Внимание: приложения с MDI регистрируют все серверные объекты
// вне зависимости от наличия ключей /Embedding или
// /Automation в командной строке.
```

Созданная AppWizard исходная программа после анализа командной строки проверяет, не было ли приложение запущено как внедренное (или с целью обеспечения автоматизации). Если это так, нет необходимости продолжать процесс инициализации приложения, поэтому выполняется возврат из функции, как показано в листинге 15.4.

Листинг 15.4. Фрагмент файла ShowString.cpp — проверка типа запуска приложения

```
// Проверка, не было ли приложение запущено как сервер OLE.
if (cmdInfo.m_bRunEmbedded || cmdInfo.m_bRunAutomated)
{

   // Приложение было запущено с ключами /Embedding или
   // /Automation. В этом случае главное окно не отображается.
   return TRUE;
}
```

Если приложение было запущено как отдельное самостоятельное задание, обновляется системный реестр.

```
// Когда серверное приложение запускается как отдельное
// самостоятельное задание, полезно будет обновить системный
// реестр на тот случай, если он был поврежден.
m_server.UpdateRegistry(OAT_INPLACE_SERVER);
```

Информация, связанная с ActiveX, хранится в системном реестре (Registry). (Этот реестр обсуждался нами в главе 7.) Когда пользователь выбирает команды Insert⇨Object или Edit⇨Insert Object, системный реестр предоставляет список типов объектов, которые могут быть вставлены в документы на данном компьютере. Однако, чтобы ShowString появилось в этом списке, оно должно быть зарегистрировано. Многие разработчики вводят соответствующий программный фрагмент в инсталляционную программу, автоматически регистрируя свое серверное приложение. Но MFC идет здесь на шаг дальше, выполняя регистрацию приложения всякий раз, когда оно запускается. Если файлы приложения были перемещены или изменены, соответствующая информация будет автоматически обновлена при следующей регистрации, которая будет выполнена при запуске приложения в качестве отдельного самостоятельного задания.

Класс CShowStringDoc

Класс документа CShowStringDoc теперь наследуется от класса COleServerDoc, а не от класса CDocument. Соответственно в начало файла showstring.cpp добавлена следующая строка:

```
#include "SrvrItem.h"
```

Этот файл заголовка содержит описание класса серверного объекта CShowStringSrvrItem, обсуждаемого в соответствующем подразделе данного раздела. В конструктор класса CShowStringDoc::CShowStringDoc() добавлена следующая строка:

```
EnableCompoundFile();
```

Она подключает средства работы с составными файлами.

В тексте файла заголовка появилась новая открытая функция, так что теперь прочие функции получили доступ к экземпляру объекта сервера:

```
CShowStringSrvrItem* GetEmbeddedItem()
   { return (CShowStringSrvrItem*)COleServerDoc::GetEmbeddedItem(); }
```

Здесь выполняется вызов метода базового класса GetEmbeddedItem(), который, в свою очередь, вызывает функцию OnGetEmbeddedItem(). Этот метод должен быть переопределен в классе документа ShowString, как показано в листинге 15.5.

Листинг 15.5. Файл ShowStringDoc.cpp — метод CShowStringDoc::OnGetEmbeddedItem()

```
COleServerItem* CShowStringDoc::OnGetEmbeddedItem()
{

   // Функция OnGetEmbeddedItem вызывается базовой подпрограммой
   // для того, чтобы получить доступ к объекту класса
   // COleServerItem, который связан с документом.
   // Она вызывается только при необходимости.

   CShowStringSrvrItem* pItem = new CShowStringSrvrItem(this);
   ASSERT_VALID(pItem);
   return pItem;
}
```

Этот фрагмент программы создает для данного документа новый серверный экземпляр объекта и возвращает указатель на него.

Класс CShowStringView

В классе представления добавлен новый элемент в карту сообщений:

```
ON_COMMAND(ID_CANCEL_EDIT_SRVR, OnCancelEditSrvr)
```

Теперь при перехвате сообщения ID_CANCEL_EDIT будет выполняться отказ от редактирования на месте. Для этого сообщения уже был определен акселератор — клавиша <Esc>. Функция, которая выполняет обработку данного сообщения, выглядит следующим образом:

```
void CShowStringView::OnCancelEditSrvr()
{
   GetDocument()->OnDeactivateUI(FALSE);
}
```

Данная функция просто деактивизирует экземпляр объекта. Никаких других изменений в представлении не производится — представления серверов намного проще представлений контейнеров.

Класс CShowStringSrvrItem

Этот класс экземпляра объекта-сервера является в приложении ShowString совершенно новым элементом. Он обеспечивает интерфейс между контейнерным приложением, вызвавшим запуск ShowString, и документом ShowString в этом приложении. Данный класс описывает весь документ ShowString, который внедрен в другой документ, или же часть документа

ShowString, которая связана с контейнерным документом. Класс не имеет переменных-членов, кроме наследованных от базового класса COleServerItem. В данном классе переопределяются восемь методов.

- Конструктор
- Деструктор
- GetDocument()
- AssertValid()
- Dump()
- Serialize()
- OnDraw()
- OnGetExtent()

Конструктор просто передает указатель на документ в базовый класс. Деструктор не выполняет никаких действий. Метод GetDocument() является передаточной функцией, которая вызывает метод базового класса с тем же именем и анализирует возвращаемое значение. Методы AsserValid() и Dump() являются отладочными функциями, которые вызывают функции базового класса. Метод Serialize() выполняет некоторые реальные действия, как показано в листинге 15.6.

Листинг 15.6. Файл SrvrItem.cpp — метод CShowStringSrvrItem::Serialize()

```
void CShowStringSrvrItem::Serialize(CArchive& ar)
{

   // Функция CShowStringSrvrItem::Serialize вызывается
   // управляющей программой, если экземпляр объекта
   // копируется в Clipboard. Это может происходить
   // автоматически, посредством обратного вызова OLE функции
   // OnGetClipboardData(). Для внедренных объектов по
   // умолчанию лучше всего просто делегировать
   // документ функции Serialize(). Если приложение
   // поддерживает связывание, то вам потребуется упорядочивать
   // части документа.

   if (!IsLinkedItem())
   {
      CShowStringDoc* pDoc = GetDocument();
      ASSERT_VALID(pDoc);
      pDoc->Serialize(ar);
   }
}
```

Здесь нет необходимости дублировать работу. Если экземпляр объекта является внедренным, то он является полным документом, а такой документ имеет вполне работоспособную функцию Serialize(), которая может сама справиться с задачей. В автоматически генерируемый текст AppWizard не включает средств для упорядочения связанных экземпляров объектов, поскольку эта процедура зависит от специфики приложения. Вам здесь следовало бы сохранять информацию, достаточную для описания того, какая часть документа связана. Например, в случае электронной таблицы это могут быть ячейки от A3 до D27. Все это не имеет смысла для приложения ShowString, поэтому не нужно вносить в функцию Serialize() никаких изменений.

Может показаться, что функция OnDraw() находится здесь не на своем месте. И это вполне логичная мысль в отношении функции представления. Но данная функция OnDraw() рисует

изображение экземпляра серверного объекта в случае, если он является неактивным. Будучи активным, объект должен быть очень похож на представление, и имеет смысл разделить работу между функциями CShowStringView::OnDraw() и CShowStringSrvrItem::OnDraw(). Текст последней из этих функций, автоматически сгенерированный мастером AppWizard, приведен в листинге 15.7.

Листинг 15.7. Файл SrvrItem.cpp — метод CShowStringSrvrItem::OnDraw()

```
BOOL CShowStringSrvrItem::OnDraw(CDC* pDC, CSize& rSize)
{
    CShowStringDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);


    // TODO: задать размер и тип привязки. (Размер обычно
    // равен размеру, возвращаемому функцией OnGetExtent().)
    pDC->SetMapMode(MM_ANISOTROPIC);
    pDC->SetWindowOrg(0,0);
    pDC->SetWindowExt(3000, 3000);


    // TODO: добавьте сюда текст программы,
    // реализующей построение изображения. Возможно, заполните
    // прямоугольник, для которого указаны относительные размеры.
    // Все графические построения должны выполняться в контексте
    // устройства метафайла (pDC).

    return TRUE;
}
```

Внесение всех предложенных изменений потребует немалого труда, но в данном случае это не нужно, поскольку, в отличие от функции CShowStringView::OnDraw(), данная функция имеет два аргумента. Второй аргумент — размеры прямоугольника, в котором формируется изображение неактивного объекта. Границы же, как отмечено в примечаниях к тексту функции, обычно предоставляются функцией OnGetExtent(), текст которой показан в листинге 15.8.

Листинг 15.8. Файл SrvrItem.cpp — метод CShowStringSrvrItem:: OnGetExtent()

```
BOOL CShowStringSrvrItem::OnGetExtent(DVASPECT dwDrawAspect, CSize& rSize)
{

    // Большинство приложений, как и это, отрабатывают только
    // процесс рисования содержимого объекта. Если вам необходимо
    // обеспечить поддержку и других аспектов, как, например,
    // DVASPECT_THUMBNAIL (переопределяя функцию OnDrawEX), то
    // вызов OnGetExtent должен быть изменен
    // с целью обработки дополнительных аспектов.

    if (dwDrawAspect != DVASPECT_CONTENT)
        return COleServerItem::OnGetExtent(dwDrawAspect, rSize);


    // Функция CShowStringSrvrItem::OnGetExtent вызывается для
    // получения параметров границ в единицах HIMETRIC для всего экземпляра
    // объекта. Текст, автоматически размещенный здесь, просто
    // возвращает жестко заданное количество единиц.

    CShowStringDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
```

```
    // TODO: заменить данную строку реальными размерами

    rSize = CSize(3000, 3000); // 3000x3000 единиц HIMETRIC.

    return TRUE;
}
```

Очень скоро мы внесем в эту функцию все необходимые изменения.

Класс CInPlaceFrame

Класс фрейма редактирования на месте, наследуемый от класса COleIPFrameWnd, управляет рамкой, помещаемой вокруг серверного экземпляра объекта, а также отображаемыми в ней панелями инструментов, строкой состояния и панелями диалоговых окон, в совокупности называемых *панелями управления*. Класс включает в себя три следующих защищенных переменных-члена:

```
CToolBar m_wndToolBar;
COleResizeBar m_wndResizeBar;
COleDropTarget m_dropTarget;
```

Класс CToolBar обсуждался нами в главе 9. Класс COleDropTarget обсуждался в разделе главы 14, посвященном программированию операции *перетащить и опустить*. Класс COleResizeBar очень похож на класс CRectTracker, который обсуждался в главе 14, но в отличие от последнего он предоставляет возможность изменять размеры серверного экземпляра объекта, а не контейнерного.

Ниже перечислены семь методов класса CInPlaceFrame.

- Конструктор
- Деструктор
- AssertValid()
- Dump()
- OnCreate()
- OnCreateControlBars()
- PreCreateWindow()

Конструктор и деструктор не выполняют никаких действий. Методы AssertValid() и Dump() являются отладочными, они просто вызывают методы базового класса. Метод OnCreate() включает нетривиальные операторы, представленные в листинге 15.9.

Листинг 15.9. Файл IPFrame.cpp — метод CInPlaceFrame::OnCreate()

```
int CInPlaceFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (COleIPFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;


    // Класс CResizeBar обеспечивает изменение размеров на месте.
    if (!m_wndResizeBar.Create(this))
    {
        TRACE0("Failed to create resize bar\n");
        return -1; // fail to create
    }
```

```
    // По умолчанию предусматривается регистрация
    // данного объекта.
    // Это обеспечит защиту от возможного "проваливания"
    // перетаскиваемого объекта.
    m_dropTarget.Register(this);

    return 0;
}
```

Метод перехватывает сообщение WM_CREATE, которое посылается, когда создается и выводится на экран рамка редактирования на месте. Он вызывает метод базового класса, а затем создает фрейм с изменяемыми размерами. После этого объект регистрируется в качестве адресата перетаскивания; если что-либо будет в процессе перетаскивания помещено в данный фрейм редактирования на месте, то объект будет считаться помещенным в сервер, а не в контейнер, находящийся *под* ним.

Когда документ сервера активизируется на месте, метод COleServerDoc::ActivateInPlace() вызывает метод CInPlaceFrame::OnCreateControlBars(), текст которого представлен в листинге 15.10.

Листинг 15.10. Файл IPFrame.cpp — метод CInPlaceFrame::OnCreateControlBars()

```
BOOL CInPlaceFrame::OnCreateControlBars(CFrameWnd* pWndFrame, CFrameWnd* pWndDoc)
{

    // Устанавливает владельца этого окна, так что сообщения
    // будут передаваться по назначению.
    m_wndToolBar.SetOwner(this);


    // Панель инструментов создается в фрейме окна клиента.
    if (!m_wndToolBar.Create(pWndFrame) ||
        !m_wndToolBar.LoadToolBar(IDR_SHOWSTTYPE_SRVR_IP))
    {
        TRACE0("Failed to create toolbar\n");
        return FALSE;
    }


    // TODO: удалите данный фрагмент, если в приложении
    // не используются контекстные окна указателя и
    // плавающие панели инструментов.
    m_wndToolBar.SetBarStyle(m_wndToolBar.GetBarStyle() |
        CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC);


    // TODO: удалите эти три строки, если не предусматривается
    // использование стационарных панелей инструментов.
    m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
    pWndFrame->EnableDocking(CBRS_ALIGN_ANY);
    pWndFrame->DockControlBar(&m_wndToolBar);


    return TRUE;

}
```

Метод создает стационарную панель инструментов с регулируемыми размерами, для пиктограмм которой будут выводиться контекстные окна указателя. В исходном состоянии панель инструментов располагается вдоль края рамки главного окна приложения.



Если вы разрабатываете приложение с многодокументным интерфейсом и хотите расположить панель инструментов вдоль рамки окна документа, используйте при вызове функции m_wndToolBar.Create() указатель pWndDoc вместо указателя pWndFrame, но предварительно убедитесь, что он не равен NULL.

Последним методом в классе CInPlaceFrame является PreCreateWindow(). В данный момент он просто вызывает одноименный метод базового класса, как показано в листинге 15.11.

Листинг 15.11. Файл IPFrame.cpp — метод CInPlaceFrame::PreCreateWindow()

```
BOOL CInPlaceFrame::PreCreateWindow(CREATESTRUCT& cs)
{

   // TODO: модифицировать класс окна или стили, внеся
   // изменения в структуру CREATESTRUCT cs.

   return COleIPFrameWnd::PreCreateWindow(cs);
}
```

Этот метод вызывается перед вызовом OnCreate() и устанавливает стили для фрейма окна посредством структуры CREATESTRUCT.

Внимание!

Модифицирование этих стилей нельзя проводить по собственному усмотрению. В документации Microsoft рекомендуется предварительно прочитать исходные тексты для всех классов в иерархии класса CInPlaceFrame (CWnd, CFrameWnd, COleIPFrameWnd), чтобы увидеть, какие элементы CREATESTRUCT уже определены, прежде чем вносить какие-либо изменения. Для нашего приложения не вносите изменений в CREATESTRUCT.

Недоработки, имеющиеся в данном сервере

Помимо того, что исходное приложение, сгенерированное AppWizard, не выводит на экран строки, какие еще функции сервера в нем опущены? Указания // TODO: в методах OnDraw() и GetExtent() только отмечают задания, выполнение которых предоставлено мастером AppWizard лично вам. Попробуйте оттранслировать приложение SnowString, а затем запустите его как самостоятельное приложение просто для того, чтобы проверить регистрацию.

На рис. 15.4 показано диалоговое окно Object приложения Microsoft Word, которое можно вывести на экран, выбрав команду Insert⇨Object. SnowString представлено в списке строкой SnowSt Document, что не удивительно, если вспомнить, что его меню был присвоен идентификатор IDR_SHOWSTTYPE. Среда разработки Visual Studio присвоила нашему документу имя ShowSt document. Это значение вполне возможно переопределить непосредственно в процессе настройки AppWizard, щелкнув на кнопке Advanced на этапе 4. На рис. 15.5 показано соответствующее диалоговое окно, а также длинное и короткое имена типа файла.

Таким образом, названия типа файла, помещенные в системный реестр, были установлены для нашего проекта некорректно. На нескольких последующих страницах этой книги вам будет предложен небольшой обзор методов хранения имен типов файлов документов, а также показано, как сложно их изменять.

Имя типа файла хранится в таблице строк. Имя — это значение ресурса IDR_SHOWSTTYPE, и мастер AppWizard формирует его в следующем виде:

```
\nShowSt\nShowSt\n\n\nShowString.Document\nShowSt Document
```

Чтобы увидеть эту строку, на вкладке ResourseView выберите элемент String Table и откройте имеющуюся в нем единственную таблицу строк. В окне таблицы щелкните на строке IDR_SHOWSTTYPE с тем, чтобы выделить ее, а затем выберите команду View⇒Properties. При создании нового экземпляра документа данная строка помещается в его шаблон функцией CShowStringApp::InitInstance() так, как показано в листинге 15.12.

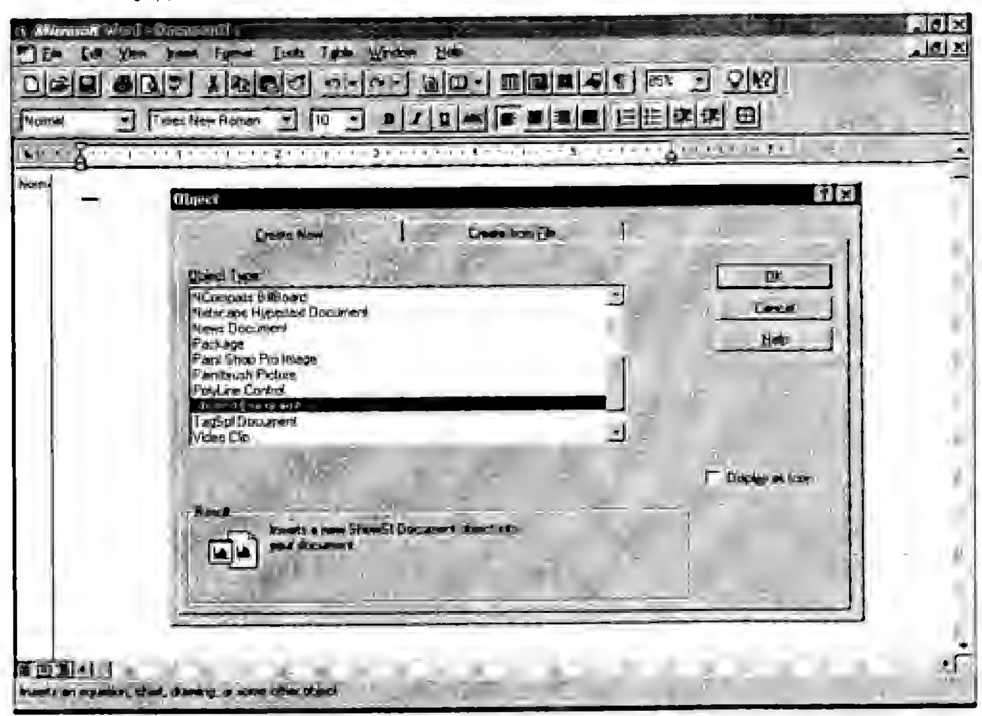


Рис. 15.4. Теперь в диалоговом окне Object при вставке новых объектов в документ Word присутствует тип документов ShowString, называемый ShowSt Document

Листинг 15.12. Файл ShowString.cpp — фрагмент функции ShowStringApp::InitInstance()

```
pDocTemplate = new CMultiDocTemplate(
    IDR_SHOWSTTYPE,
    RUNTIME_CLASS(CShowStringDoc),
    RUNTIME_CLASS(CChildFrame), // Рамка дочернего окна MDI.
    RUNTIME_CLASS(CShowStringView));
```

Заголовок меню ресурсов состоит из семи строк, и каждая из них используется в различных частях фрейма окна. Строки разделяются символами перевода строки \n. Ниже будет описано назначение каждой из этих семи строк и на примере приложения ShowString показано, какие значения устанавливает для них по умолчанию мастер AppWizard.

- **Заголовок окна** используется SDI-приложениями для вывода в строку заголовка. В приложении ShowString не используется.

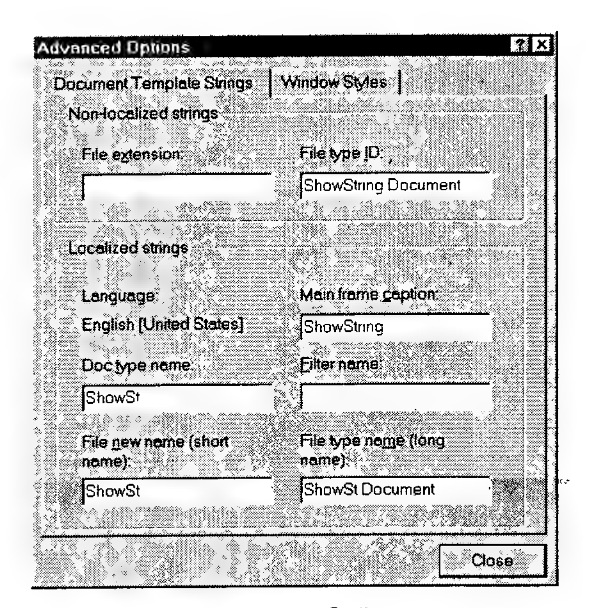


Рис. 15.5. В окне Advanced Options *на четвертом этапе настройки мастера* `AppWizard` *можно изменить принятое по умолчанию имя типа файла*

- **Имя документа** используется в качестве корня для построения имен документов по умолчанию. В приложении `ShowString` имеет значение `ShowSt`, поэтому новым документам будут присваиваться имена `ShowSt1`, `ShowSt2` и т.д.
- **Имя нового файла** будет предложено для файлов данного типа в диалоговом окне File New. (К примеру, в Visual Studio имеется восемь типов файлов, включая типы `Text File` (текстовый файл) и `Project Workspace` (файл среды проекта).) Для `ShowString` имеет значение `ShowSt`.
- **Имя фильтра** — элемент раскрывающегося списка List File of Type (список файлов по типам) в диалоговом окне File Open. Для `ShowString` опущено.
- **Расширение фильтра** — расширение, которое соответствует имени фильтра. Для `ShowString` опущено.
- **Идентификатор типа файла в реестре** — короткая строка, которая будет помещена в системный реестр. Для `ShowString` имеет значение `ShowString.Document`.
- **Имя типа файла в реестре** — длинная строка, которая выводится в диалоговых окнах, обращающихся за информацией в реестр. Для `ShowString` имеет значение `ShowSt Document`.

(Еще раз взгляните на рис. 15.5, и вы поймете, откуда взяты все приведенные выше значения.) Попробуйте изменить последнее значение в строке. В диалоговом окне Properties измените текст в поле Caption так, чтобы последний элемент строки получил значение `ShowString Document`. Оттранслируйте приложение и запустите его на выполнение, после чего сразу же закройте. В окне вывода сообщения Visual Studio вы увидите следующие сообщения.

```
Warning: Leaving value ShowSt Document for key
   ShowString.Document in registry
   intended value was ShowString Document.
```

```
Warning: Leaving value ShowSt Document for key
   CLSID\{0B1DEE40-C373-11CF-870C-00201801DDD6} in registry
   intended value was ShowString Document.
```

```
Предупреждение: для ключа ShowString.Document в Registry
   существующее значение ShowSt Document,
   предлагается новое значение ShowString Document

Предупреждение: для ключа CLSID\{0B1DEE40-C373-11CF-870C-00201801DDD6}
   в Registry существующее значение ShowSt Document,
   предлагается новое значение ShowString Document.
```

Это означает, что при вызове функции UpdateRegistry() обновление этих двух ключей выполнено не было. Можно, используя аргументы функции UpdateRegistry(), потребовать обязательного обновления указанных ключей, но этот путь намного сложнее, чем тот, который предлагается ниже. Поскольку никаких изменений в программы, сгенерированные AppWizard, мы еще не вносили, проще всего будет удалить папку ShowString вместе со всем ее содержимым и создать это приложение заново, но на этот раз уже задав требуемое длинное имя ShowString для документов.

Внимание!

Всегда проверяйте сгенерированный AppWizard текст программы, прежде чем приступить к внесению изменений. Пока вы не будете свободно ориентироваться во всех значениях по умолчанию, которые вам предлагаются, имеет смысл потратить немного времени на анализ того, что вы получили, прежде чем двигаться дальше. Перезапустить AppWizard очень легко, но, если вы уже потратили несколько часов на внесение изменений, а затем выясняется, что необходимо все начать сначала, вряд ли эта простота вас порадует.

Удалите папку ShowString со всем ее содержимым и сгенерируйте с помощью AppWizard новое приложение, такое же, как и прежде. На этот раз на этапе 4 настройки щелкните на кнопке Advanced и внесите изменения в имена типа файла — поля File new name (short name) и File type name (long name). Закончив изменения, щелкните на кнопке Finish. Мастер AppWizard выведет окно с вопросом, желаете ли вы использовать для приложения уже существующий CLSID. Щелкните в этом окне на кнопке Yes, а затем для создания проекта — на кнопке OK. В результате всех этих манипуляций будет создан новый файл showstring.reg, содержащий значения, предназначенные для внесения в системный реестр Registry.

Выполненные изменения были внесены как в таблицу строк, так и в файл showstring.reg. Теперь можно ожидать, что после повторной трансляции и запуска приложения ошибка будет, наконец, исправлена. И это действительно так — когда вы запустите приложение, оно потребует выполнить необходимое нам обновление реестра Registry, используя значения из новой таблицы строк. Тем не менее обновление записей в системном реестре не будет осуществлено и на этот раз. Если вы выполните все указанные выше операции, в окне выходной информации появятся следующие сообщения.

```
Warning: Leaving value ShowSt Document for key
   ShowString.Document in registry
   intended value was ShowString Document.

Warning: Leaving value ShowSt Document for key
   CLSID\{0B1DEE40-C373-11CF-870C-00201801DDD6} in registry
   intended value was ShowString Document

Warning: Leaving value ShowSt for key
   CLSID\{0B1DEE40-C373-11CF-870C-00201801DDD6}\AuxUserType\2
```

```
in registry
intended value was ShowString.
```

```
Предупреждение: для ключа CLSID\{0B1DEE40-C373-11CF-870C-00201801DDD6}
   в Registry сохранено значение ShowSt Document,
   предлагается новое значение ShowString Document

Предупреждение: для ключа ShowString.Document в Registry
   сохранено значение ShowSt Document,
   предлагается новое значение ShowString Document

Предупреждение: для ключа CLSID\{0B1DEE40-C373-11CF-870C-00201801DDD6}\AuxUserType\2
   в Registry сохранено значение ShowSt,
   предлагается новое значение ShowString.
```

Итак, как же нам теперь выбраться из этого замкнутого круга? Необходимо отредактировать сам системный реестр — Registry. Если для вас это звучит устрашающе, то так и должно быть. Повредив этот файл, можно сделать систему совершенно неработоспособной. Однако мы не собирается вносить в нее изменения вручную и корректировать значения содержащихся в ней ключей. Наоборот, мы планируем использовать файл реестра, который был подготовлен мастером AppWizard. Для этого выполните следующие действия.

1. На панели задач Windows 95 выберите команду Start⇨Run (Пуск⇨Выполнить).
2. В поле ввода раскрывшегося окна введите regedit и нажмите клавишу <Enter>.
3. В окне Registry Editor (Редактор реестра) выберите команду Registry⇨Import Registry File (Реестр⇨Импорт файла реестра).
4. В окне Import Registry File (Импорт файла реестра) пролистайте папки, хранящиеся на жестком диске, и перейдите к той, в которой мастером AppWizard только что было создано новое приложение-сервер ShowString (рис. 15.6). Щелкните на кнопке Open (Открыть).
5. На экран будет выведено сообщение об успешном завершении работы. Щелкните на кнопке OK.
6. Закройте окно приложения Registry Editor.

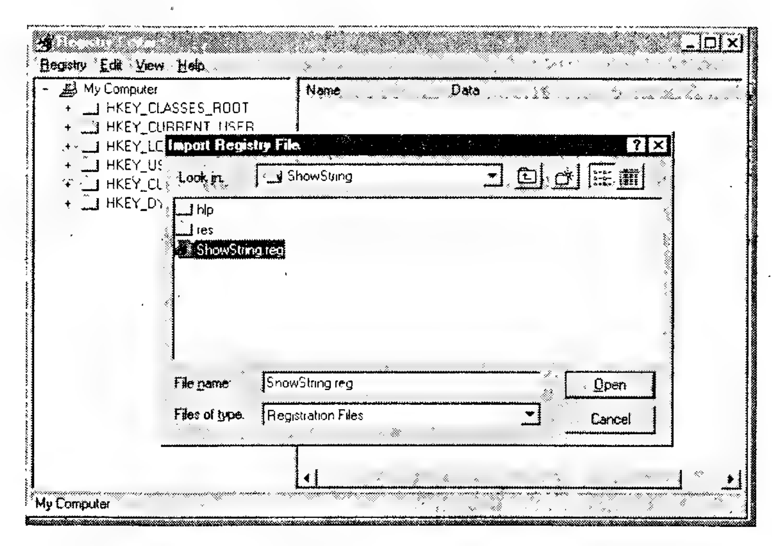


Рис. 15.6. Файлы системного реестра, сгенерированные мастером AppWizard, имеют расширение .reg

Теперь, если снова запустить приложение ShowString, указанные выше сообщения об ошибках не появятся. Запустите Word и выберите команду Insert⇨Object. В данном случае диалоговое окно Object содержит более осмысленное название для документов ShowString, как показано на рис. 15.7.

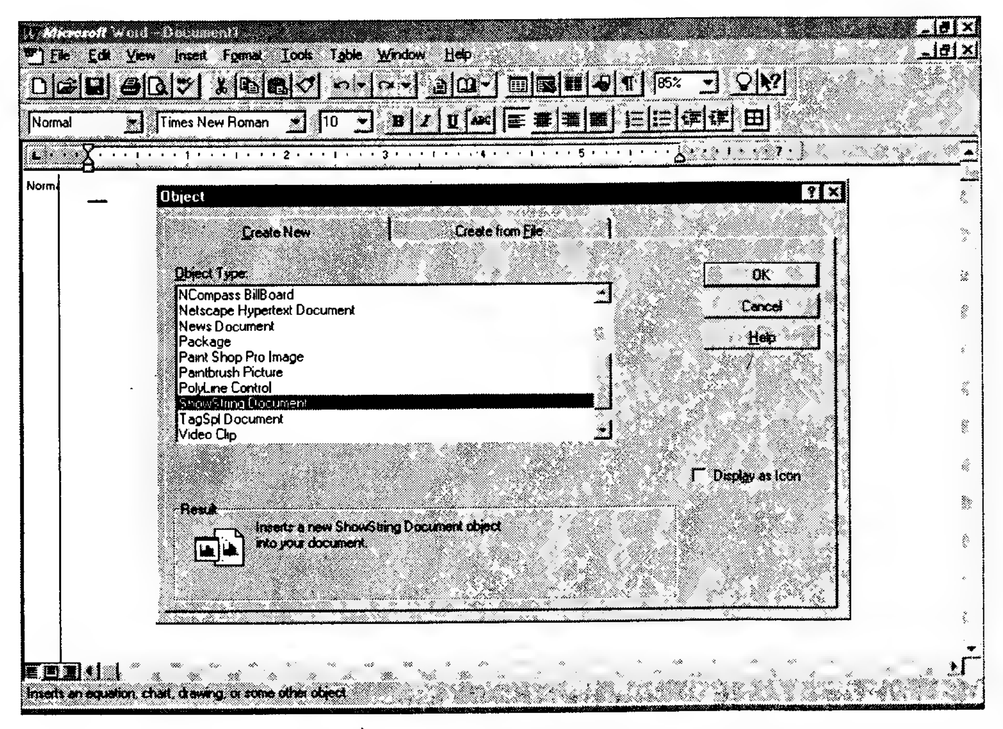


Рис. 15.7. В списке диалогового окна Object *среди прочих приложений появилось обновленное длинное имя типа файла*

На заметку

Из всвго изложвнного выше можно сделать три вывода. Во-первых, следует очень хорошо подумать, прежде чем сделать щелчок на кнопке Finish в диалоговом окне AppWizard. Во-вторых, не следует игнорировать системный реестр, если вы разрабатываете приложение ActiveX. В-третьих, изменить можно все, что угодно, если у вас на это хватит терпения.

Для вставки объекта ShowString в документ Word щелкните в диалоговом окне Object на кнопке OK. Вам немедленно будет предоставлена возможность отредактировать внедренный объект на месте, как показано на рис. 15.8. Обратите внимание, что в окне Word используется комбинированное меню редактирования на месте, составленное из меню сервера и контейнера. В данный момент с внедренным объектом ShowString можно проделать очень немногое, поскольку собственно текст программы ShowString, выводящий на экран строку, в приложение еще не внесен. Для завершения редактирования на месте нажмите клавишу <Esc>. Меню в окне Word приобретут свой обычный вид, как показано на рис. 15.9.

Хотя данный сервер еще не делает ничего конкретного, функции сервера он выполняет очень хорошо. У вас уже есть возможность изменять размеры и перемещать внедренный экземпляр объекта, когда он является активным или неактивным, причем все работает именно так, как вы того ожидаете. Все, что нам осталось сделать, — это восстановить функциональные возможности собственно приложения ShowString.

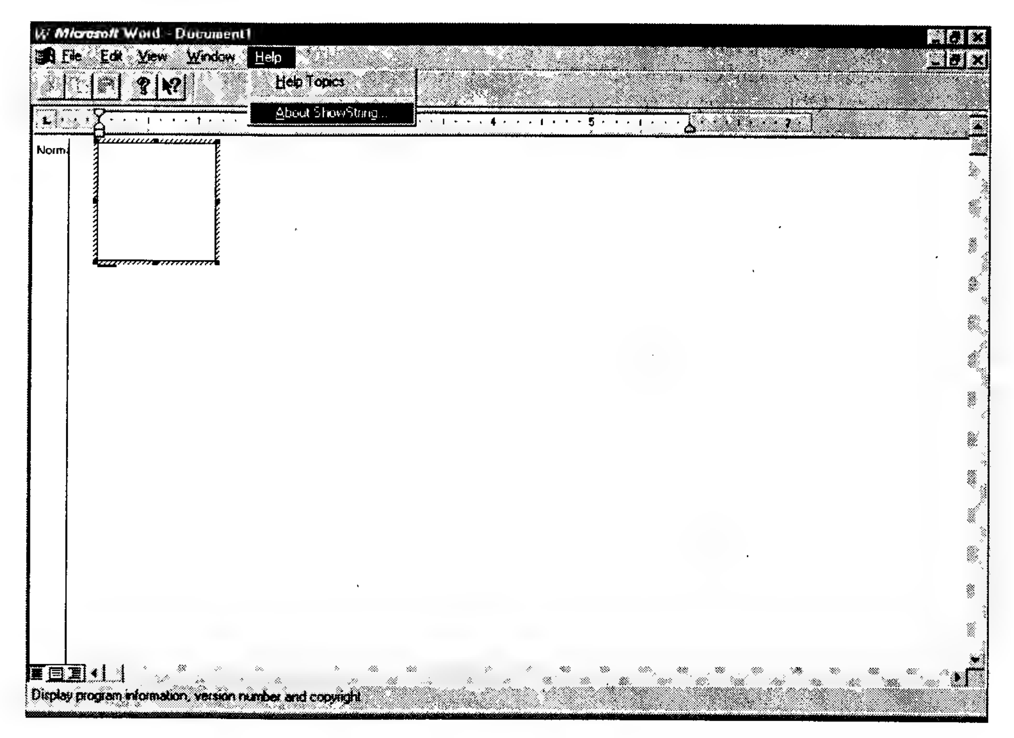


Рис. 15.8. При редактировании на месте меню Word заменяются комбинированным меню сервера и контейнера

Восстановление функциональных возможностей ShowString

Как и в предыдущей главе, мы подошли к моменту, когда необходимо добавить в созданное и исследованное нами приложение-сервер функциональные возможности ShowString. Если в предыдущий раз вы выполнили все, что от вас требовалось, то на этот раз подобная процедура потребует меньше времени. Не забудьте открыть файлы приложения ShowString, созданные в главе 8, что даст возможность копировать текст программы и ресурсы из работающего обычного приложения в только что созданную заготовку сервера ActiveX. Вот что необходимо выполнить.

1. В файле ShowStringDoc.h добавьте в определение класса закрытые переменные-члены и открытый метод Get.
2. В функцию CShowStringDoc::Serialize() вставьте операторы сохранения и восстановления этих переменных-членов.
3. В функцию CShowStringDoc::OnNewDocument() вставьте текст программы, в котором будет выполняться инициализация этих переменных.
4. Скопируйте из исходного ShowString в новое приложение-сервер все меню Tools, для чего выберите команду File⇨Open и откройте прежний файл ShowString.rc. Затем откройте меню IDR_SHOWSTTYPE, выделите меню Tools и выберите команду Edit⇨Copy. Далее откройте меню IDR_SHOWSTTYPE нового ShowString, выделите меню Window и выберите команду Edit⇨Paste.

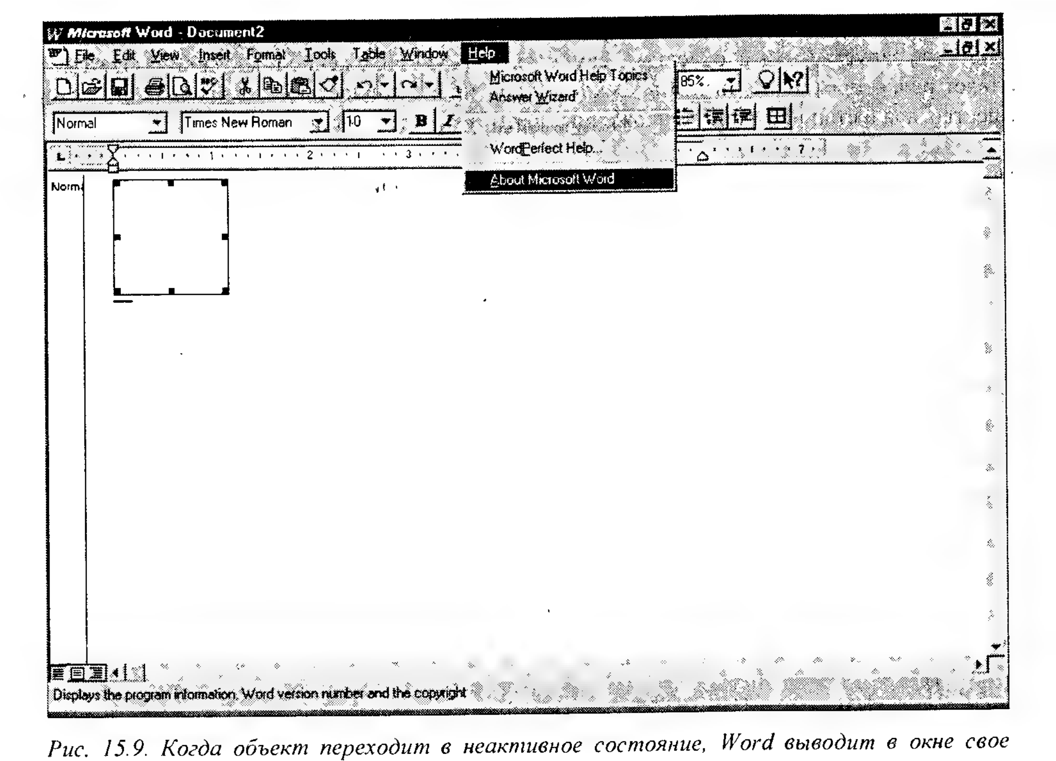


Рис. 15.9. Когда объект переходит в неактивное состояние, Word выводит в окне свое обычное меню

5. Тем же самым способом вставьте меню **Tools** в меню IDR_SHOWSTTYPE_SRVR_IP (перед строками разделителей) и в меню IDR_SHOWSTTYPE_SRVR_EMB.
6. Добавьте акселератор <Ctrl+T> для команды ID_TOOLS_OPTIONS так, как это описано в главе 8. Добавление выполните во всех трех меню.
7. Удалите из нового приложения-сервера диалоговое окно IDD_ABOUTBOX. Скопируйте диалоговые окна IDD_ABOUTBOX и IDD_OPTIONS из старого ShowString в новое.
8. Пока фокус находится на ресурсе IDD_OPTIONS, выберите команду **View⇨Class Wizard**. Создайте класс COptionsDialog, аналогичный созданному в исходном ShowString.
9. Используйте Classwizard для организации перехвата команды ID_TOOLS_OPTIONS в классе CShowStringDoc.
10. В файле ShowStringDoc.cpp замените созданную мастером AppWizard версию функции CShowStringDoc::OnToolsOptions() собственной версией, выводящей диалоговое окно.
11. В файле ShowStringDoc.cpp добавьте после существующих директив #include новую:

```
#include "OptionsDialog.h"
```

12. С помощью Classwizard подключите элементы управления диалогового окна к переменным-членам класса COptionsDialog так, как мы это делали ранее. Подключите IDC_OPTIONS_BLACK к m_color, IDC_OPTIONS_HORIZCENTER к m_horizcenter, IDC_OPTIONS_STRING к m_string и IDC_OPTIONS_VERTCENTER к m_vertcenter.

Мы не выполнили восстановление функции CShowStringView::OnDraw(), поскольку в новом приложении будут присутствовать две функции OnDraw(). Первая, показанная в листин-

ге 15.13, будет принадлежать классу представления. Она выводит строку в том случае, когда ShowString запускается как отдельное самостоятельное приложение и когда пользователь выполняет редактирование объекта на месте. Эта функция полностью аналогична той, которая существовала в старой версии приложения, поэтому просто скопируйте ее.

Листинг 15.13. Файл ShowStringView.cpp — метод CShowStringView::OnDraw()

```
void CShowStringView::OnDraw(CDC* pDC)
{
   CShowStringDoc* pDoc = GetDocument();
   ASSERT_VALID(pDoc);

   COLORREF oldcolor;
   switch (pDoc->GetColor())
   {
   case 0:
      oldcolor = pDC->SetTextColor(RGB(0,0,0)); //Черный.
      break;
   case 1:
      oldcolor = pDC->SetTextColor(RGB(0xFF,0,0)); //Красный.
      break;
   case 2:
      oldcolor = pDC->SetTextColor(RGB(0,0xFF,0)); //Зеленый.
   break;
   }

   int DTflags = 0;
   if (pDoc->GetHorizcenter())
   {
      DTflags |= DT_CENTER;
   }
   if (pDoc->GetVertcenter())
   {
      DTflags |= (DT_VCENTER|DT_SINGLELINE);
   }

   CRect rect;
   GetClientRect(&rect);
   pDC->DrawText(pDoc->GetString(), &rect, DTflags);
   pDC->SetTextColor(oldcolor);
}
```

Когда внедренный экземпляр объекта ShowString неактивен, он выводится на экран функцией CShowStringSrvrItem::OnDraw(), текст которой очень похож на текст функции OnDraw() класса представления. Однако, поскольку эта функция является членом класса CShowStringSrvrItem, а не класса CShowStringView, она не имеет доступа к тем же самым переменным-членам, что и функция OnDraw() представления. Поэтому, хотя в данном случае функция GetDocument() по-прежнему доступна, функция GetClientRect() не работает, так как она является методом класса представления, а не класса серверного экземпляра объекта. Вместо нее необходимо использовать некоторые функции-члены класса CDC. Все это дает прекрасную возможность отобразить объект на экране несколько иначе, чтобы напомнить пользователю о том, что данный объект не активен (листинг 15.14).

Листинг 15.14. Файл SrvrItem.cpp — метод CShowStringSrvrItem::OnDraw()

```
BOOL CShowStringSrvrItem::OnDraw(CDC* pDC, CSize& rSize)
{
   CShowStringDoc* pDoc = GetDocument();
```

```
    ASSERT_VALID(pDoc);


    // TODO: задать тип и границы для переназначения
    // (границы обычно соответствуют размеру,
    // возвращаемому функцией OnGetExtent).

    pDC->SetMapMode(MM_ANISOTROPIC);
    pDC->SetWindowOrg(0,0);
    pDC->SetWindowExt(3000, 3000);


    COLORREF oldcolor;
    switch (pDoc->GetColor())
    {
    case 0:
       oldcolor = pDC->SetTextColor(RGB(0x80,0x80,0x80)); //Серый.
       break;
    case 1:
       oldcolor = pDC->SetTextColor(RGB(0xB0,0,0)); // Приглушенный красный.
       break;
    case 2:
       oldcolor = pDC->SetTextColor(RGB(0,0xB0,0)); // Приглушенный зеленый.
       break;
    }

    int DTflags = 0;
    if (pDoc->GetHorizcenter())
    {
       DTflags != DT_CENTER;
    }
    if (pDoc->GetVertcenter())
    {
       DTflags != (DT_VCENTER!DT_SINGLELINE);
    }

    CRect rect;
    rect.TopLeft() = pDC->GetWindowOrg();
    rect.BottomRight() = rect.TopLeft() + pDC->GetWindowExt();
    pDC->DrawText(pDoc->GetString(), &rect, DTflags);
    pDC->SetTextColor(oldcolor);

    return TRUE;
}
```

Функция начинается текстом, автоматически сгенерированным AppWizard. Работая над приложением, которое не должно отображать свое представление на любом предложенном участке, вам потребуется добавить фрагмент, определяющий необходимые границы. Этот фрагмент заменит существующие операторы, которые устанавливают фиксированное значение(3000,3000). (Вам потребуется добавить соответствующий текст и в функцию OnGetExtent().) Но в нашем простом примере достаточно жестко заданного значения. Далее следуют операторы вывода на экран, вставленные из функции OnDraw() класса представления, но цвета, которыми отображается строка, слегка изменены, чтобы реализовать отмеченное выше напоминание пользователю.

Оттранслируйте приложение, исправив возможные мелкие ошибки, а затем запустите Excel и вставьте в его электронную таблицу документ ShowString. Новое ShowString должно работать, как прежде, выводя в центр представления строку Hello, world!. Убедитесь, что диалоговое окно Options по-прежнему работает и что вы полностью восстановили все функциональные возможности приложения. Внесите в документ хотя бы одно изменение текста

строки, цвета или центрирования, а затем нажмите клавишу <Esc> и тем самым завершите редактирование на месте. Но что это? Изображение в документе ShowString вернулось в исходное состояние — строка Hello, world! выведена в центре экранной области сервера затененной. Почему?

Вспомните, что в функции CShowStringDoc::OnToolsOptions() после щелчка пользователя на кнопке OK документу передавалось уведомление о том, что внесены изменения и необходимо выполнить перерисовку представления:

```
SetModifiedFlag();
UpdateAllViews(NULL);
```

Здесь необходимо добавить еще один оператор, обеспечивающий передачу уведомления об изменении каждому из контейнеров, содержащих данный документ:

```
NotifyChanged();
```

Заново оттранслируйте приложение и вставьте еще один объект ShowString в документ Word. На этот раз проведенные изменения будут отображены и в представлении неактивного сервера. На рис. 15.10 показан редактируемый на месте объект ShowString.

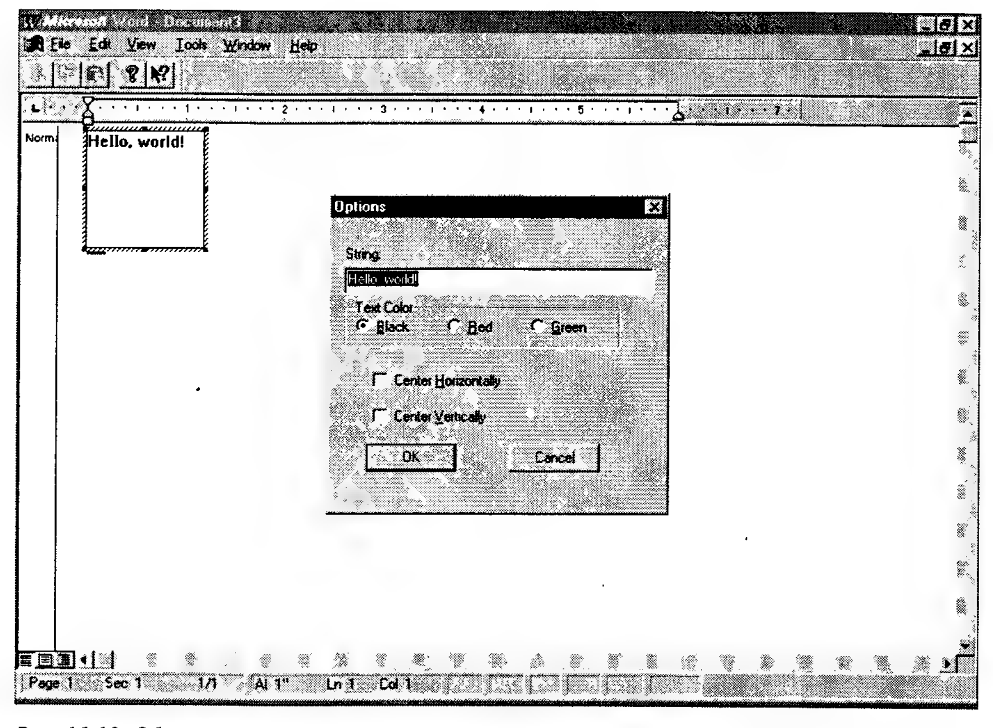


Рис. 15.10. Объект ShowString, редактируемый на месте

Наше многострадальное приложение ShowString претерпело много изменений. Пришло время для еще одной модификации.

Приложения контейнер/сервер

Как вы, вероятно, и предполагали, добавление функций контейнера ActiveX к данной версии приложения ShowString потребует столько же усилий, как и в случае описанного в предыдущей главе добавления их к обычному приложению ShowString. Если вы выполните это добавление, то в результате получите приложение, способное использовать все могущество ActiveX и предоставляющее исключительно широкие возможности при работе с документами.

Создание очередной новой версии ShowString

Чтобы преобразовать ShowString в приложение, являющееся одновременно и сервером, и контейнером, выполните следующие операции.

1. Создайте с помощью мастера AppWizard новое приложение ShowString, являющееся одновременно и контейнером, и полнофункциональным сервером. Запустите AppWizard обычным образом, но назначьте ему новую папку, отличную от тех, в которых размещаются файлы предыдущих версий ShowString. Убедитесь, что на этапе 3 вы установили переключатель Both Container And Server. На этапе 4 не забудьте сделать щелчок на кнопке Advanced и изменить полные имена типов, как это было сделано при создании предыдущей версии ShowString. И наконец, в ответ на предложение использовать прежний CLSID, ответьте No. Это совсем другое приложение.

2. Выполните все изменения, проведенные нами при создании приложения-контейнера в предыдущей главе. Затем добавьте команду Options из меню Tools и ее акселератор в главное меню приложения, меню редактирования на месте и меню редактирования внедренных объектов в отдельном окне.

3. Выполните все изменения, проведенные нами при создании приложения-сервера выше в этой главе.

4. Добавьте в новое приложение все функциональные возможности ShowString.

В данном разделе процесс построения приложения-сервера и контейнера не будет рассматриваться во всех деталях, поскольку этот материал уже был рассмотрен в предыдущей главе и в разделе *Добавление в ShowString функций сервера ActiveX* данной главы. Вместо этого мы сконцентрируем внимание на результатах, которые можно получить в результате создания такого приложения.

Вложенность и рекурсия экземпляров объектов

Благодаря тому, что приложение является одновременно и сервером (т.е. его документы могут внедряться в другие приложения), и контейнером, становится возможным создание вложенных документов. Например, электронная таблица Excel может содержать документ Word, который, в свою очередь, может содержать растровое изображение, как показано на рис. 15.11.

Работая в Excel, можно сделать двойной щелчок на документе Word и начать его редактирование на месте, как показано на рис. 15.12. Однако у вас нет возможности подобным образом продолжить работу и, сделав двойной щелчок на графическом объекте, запустить процесс редактирования на месте и для него. Вам будет предоставлена возможность редактировать его, но только в отдельном окне, как показано на рис. 15.13. ActiveX предусматривает ограничения, запрещающие пользователю запускать в приложении неограниченное количество вложенных сеансов редактирования на месте.

Рис. 15.11. Электронная таблица Excel содержит документ Word, в который внедрен графический объект

Рис. 15.12. Данный документ Word редактируется на месте

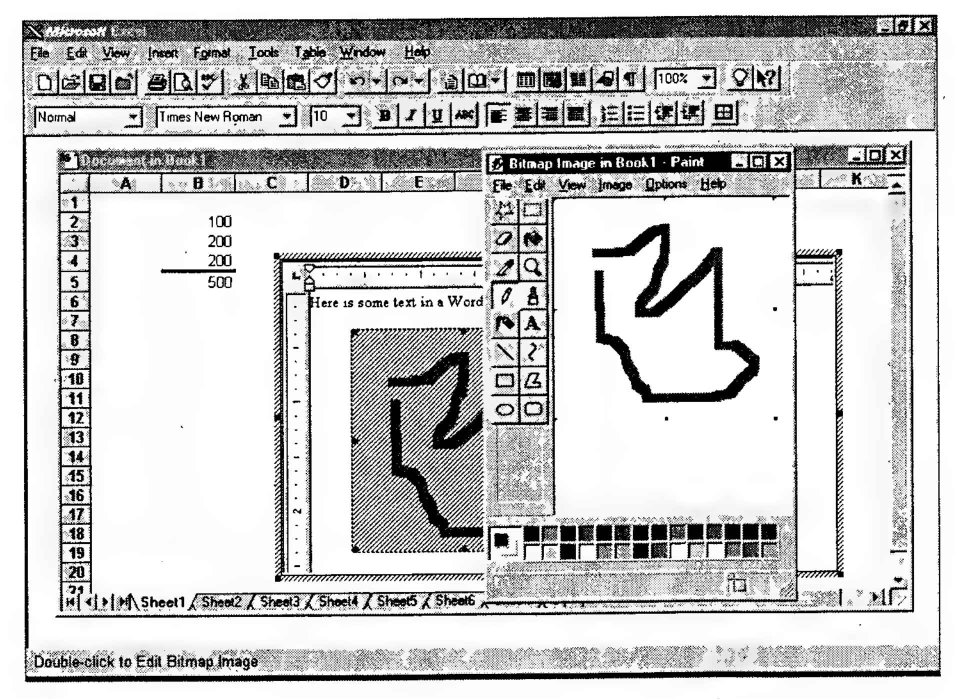


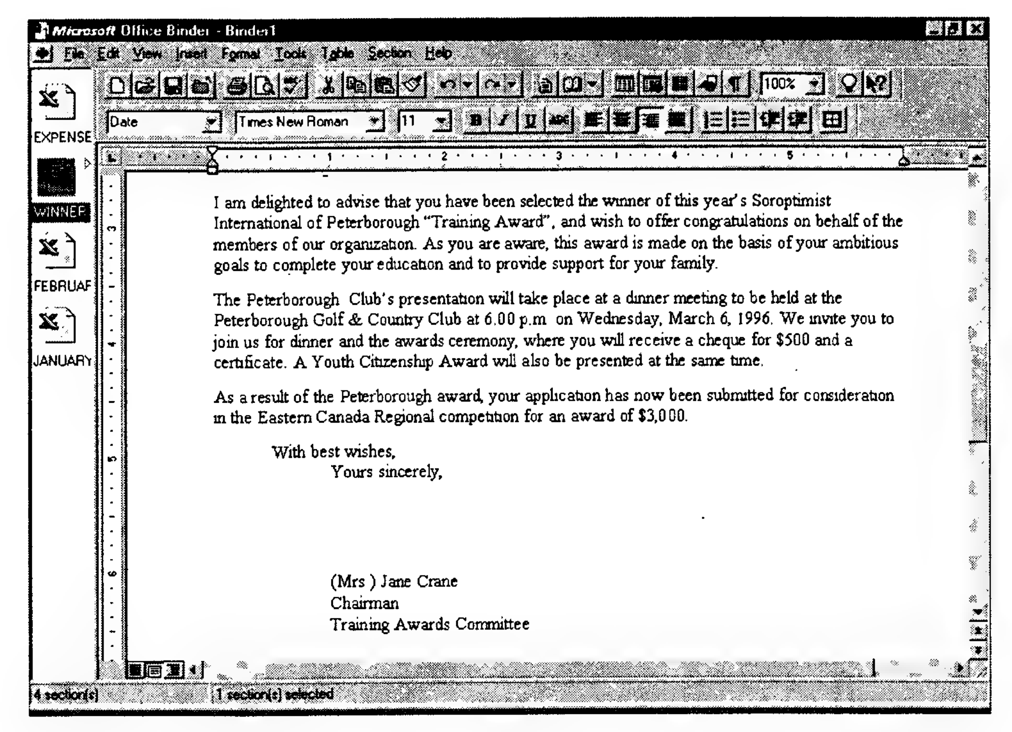
Рис. 15.13. Данный графический объект вложен в документ Word, помещенный в электронную таблицу Excel, и по этой причине не может быть отредактирован на месте. Вложенные объекты допускается редактировать только в отдельном окне

Документы ActiveX

Последним и очень важным дополнением к технологии ActiveX, сделанным совсем недавно, является концепция документов ActiveX (ActiveX documents) (иногда используется термин *объект документа ActiveX*). Обычный сервер ActiveX при редактировании документа на месте управляет меню и интерфейсом приложения-контейнера, но выполняет это в кооперации с ним. Сервер документов ActiveX идет в подобном случае намного дальше.

Что может технология документов ActiveX

Первым приложением, демонстрирующим возможности технологии документов ActiveX, стало Microsoft Office Binder, окно которого показано на рис. 15.14. У пользователя создается впечатление, что данное приложение может открыть любой из документов Office. На самом деле, хотя документы и открываются их собственными серверными приложениями, рамка вокруг них и список других документов контролируются контейнером. Microsoft Internet Explorer (версии 3.0 и более поздних) также является контейнером документов ActiveX. На рис. 15.15 показан документ Word, открытый в окне Explorer. Обратите внимание, что выведенные в окне меню являются меню Word, но сохраняется возможность использовать панель инструментов Explorer. К примеру, щелчок на кнопке Back вызовет закрытие данного документа Word и открытие предыдущего загруженного документа.



I am delighted to advise that you have been selected the winner of this year's Soroptimist International of Peterborough "Training Award", and wish to offer congratulations on behalf of the members of our organization. As you are aware, this award is made on the basis of your ambitious goals to complete your education and to provide support for your family.

The Peterborough Club's presentation will take place at a dinner meeting to be held at the Peterborough Golf & Country Club at 6.00 p.m on Wednesday, March 6, 1996. We invite you to join us for dinner and the awards ceremony, where you will receive a cheque for $500 and a certificate. A Youth Citizenship Award will also be presented at the same time.

As a result of the Peterborough award, your application has now been submitted for consideration in the Eastern Canada Regional competition for an award of $3,000.

With best wishes,
Yours sincerely,

(Mrs) Jane Crane
Chairman
Training Awards Committee

Рис. 15.14. Приложение Microsoft Office Binder упрощает объединение документов и их совместную обработку

С точки зрения пользователя, все это указывает на полный и окончательный переход к документо-ориентированной технологии. Не имеет значения, с каким приложением работает пользователь, — любой тип документов может быть открыт и отредактирован с использованием программы, написанной для работы с данным документом, причем сохраняется интерфейс, привычный для пользователя по его повседневной работе с одним и тем же приложением.

Превращение ShowString в сервер документов ActiveX

Создание еще одной версии ShowString, на этот раз в качестве сервера документов ActiveX, является совсем несложным делом. Выполните все указания раздела *Исходный вариант сервера ActiveX, формируемый AppWizard* данной главы, внеся в них два изменения. Во-первых, на третьем этапе настройки AppWizard установите флажок ActiveX document server опции поддержки документов ActiveX. Во-вторых, на четвертом этапе настройки щелкните на кнопке Advanced и в раскрывшемся диалоговом окне исправьте имена типа файла, а также внесите в поле расширения имени файла значение .sst, как показано на рис. 15.16. Это даст возможность контейнерам документов ActiveX определить, какое приложение следует запускать, если пользователь открывает файл документа ShowString.

I am delighted to advise that you have been selected the winner of this year's Soroptimist International of Peterborough "Training Award", and wish to offer congratulations on behalf of the members of our organization As you are aware, this award is made on the basis of your ambitious goals to complete your education and to provide support for your family

The Peterborough Club's presentation will take place at a dinner meeting to be held at the Peterborough Golf & Country Club at 6 00 p m on Wednesday, March 6, 1996 We invite you to join us for dinner and the awards ceremony, where you will receive a cheque for $500 and a certificate A Youth Citizenship Award will also be presented at the same time

As a result of the Peterborough award, your application has now been submitted for consideration in the Eastern Canada Regional competition for an award of $3,000

With best wishes,
Yours sincerely,

Рис. 15.15. Microsoft Internet Explorer также является контейнером документов ActiveX

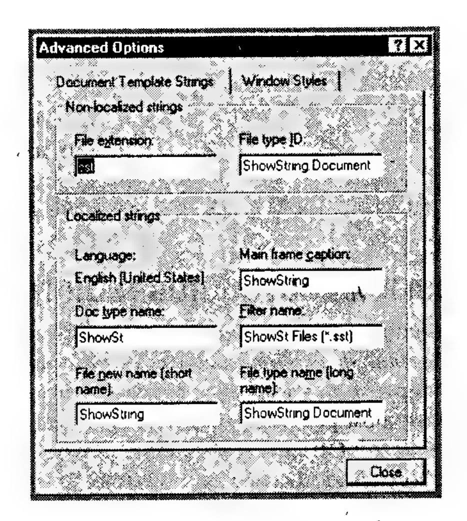


Рис. 15.16. Расширение для файлов ShowString задается в диалоговом окне Advanced Options, выводимом на четвертом этапе настройки AppWizard

Обработка расширения имени документов

Для каждой из версий ShowString, созданных нами ранее, можно было определить расширение имени файла ее документов. Когда вы задаете расширение имени файлов для приложения сервера документов ActiveX, AppWizard помещает в функцию CShowStringApp::InitInstance() следующие строки.

```
// Разрешить открытие при перетаскивании/сбросе.
m_pMainWnd->DragAcceptFiles();


// Разрешить открытие при вызове от функций DDE.
EnableShellOpen();
RegisterShellFileTypes(TRUE);
```

Существенным здесь является вызов RegisterShellFileTypes(), хотя и расширение возможностей технологии *перетащить и опустить* тоже реально. Теперь можно воспользоваться перетаскиванием файлов с рабочего стола или из папок на пиктограмму ShowString для того, чтобы запустить на выполнение новую копию этого приложения с загруженным данным документом.

Обеспечение функций сервера документов ActiveX

Установка флажка опции поддержки документов ActiveX приводит к совершенно незначительным отличиям в тексте сгенерированной мастером AppWizard программы. В функции CShowStringApp::InitInstance() из предыдущей версии ShowString, которая не являлась сервером документов ActiveX, содержался следующий вызов обновления реестра:

```
m_server.UpdateRegistry(OAT_INPLACE_SERVER);
```

Новая версия ShowString, поддерживающая технологию документов ActiveX, содержит этот оператор в таком виде:

```
m_server.UpdateRegistry(OAT_DOC_OBJECT_SERVER);
```

В обоих случаях переменная m_server имеет тип класса CShowStringSrvrItem, но в случае сервера документов ActiveX класс экземпляра серверного объекта наследуется от класса CDocObjectServerItem. Это влечет за собой множество мелких изменений по всему исходному тексту и подключаемым файлам для класса CShowStringSrvrItem везде, где вызываются функции базового класса. Аналогично класс рамки редактирования на месте CInPlaceFrame теперь наследуется от класса COleDocIPFrameWnd.

Демонстрация возможностей новейшего ShowString

Еще раз восстановите функциональные возможности собственно ShowString, как это было описано в разделе *Восстановление функциональных возможностей ShowString*, приведенном выше в этой главе. Оттранслируйте приложение, запустите его как самостоятельное приложение для регистрации в системном реестре, а затем запустите приложение Microsoft Binder (если у вас инсталлирован Microsoft Office). Для вывода на экран диалогового окна Add Section выберите команду Section⇨Add. На вкладке General выделите элемент ShowString Document и щелкните на кнопке OK.

На панели меню появится принадлежащее ShowString меню Tools. Выберите команду Tools⇨Options и измените что-нибудь. В примере на рис. 15.17 исходная строка заменена строкой Hello from the Binder (Привет от Биндера) и отключено вертикальное центрирование. Теперь в вашем распоряжении есть все функциональные возможности ShowString, хотя внешне не похоже, что вы работаете именно с этим приложением.

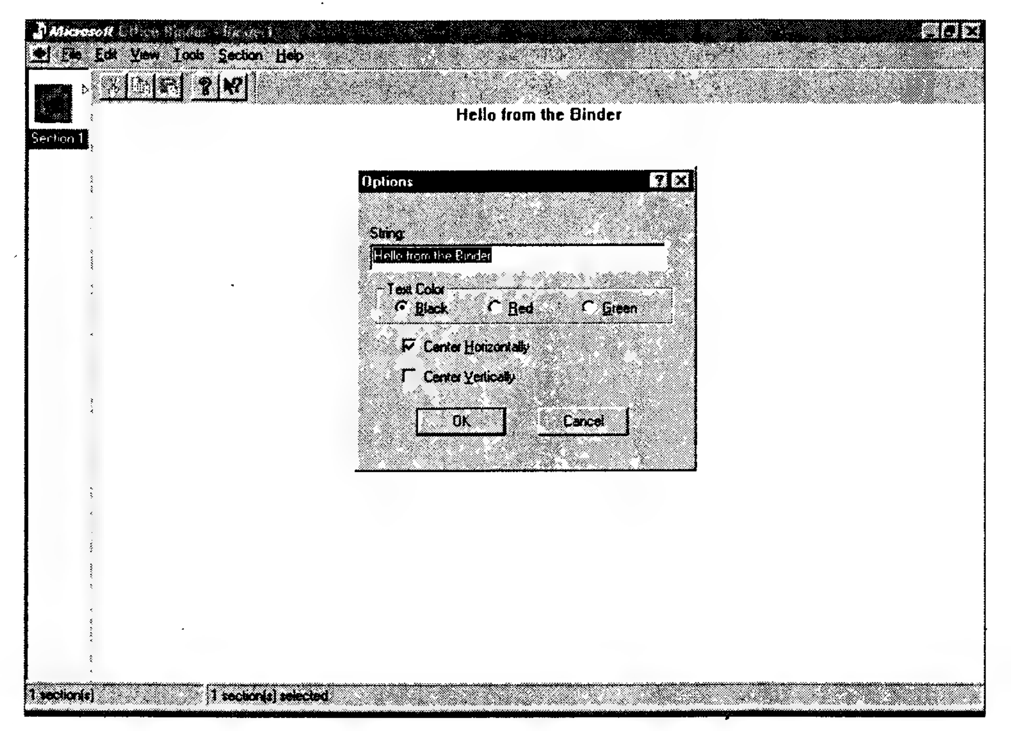


Рис. 15.17. Все функциональные возможности ShowString доступны в окне приложения Binder

А теперь запустите ShowString как самостоятельное приложение и, выбрав команду File⇨ Save, сохраните его документ. Нет необходимости вводить расширение для имени файла — расширение .sst будет добавлено автоматически. Запустите Internet Explorer 4.0 и отыщите только что сохраненный вами файл документа ShowString, а затем щелкните на кнопке Open.

Сохраненный вами документ будет открыт в окне Internet Explorer, как показано на рис. 15.18. Видно, что панель инструментов принадлежит Explorer, но в строке меню присутствует меню Tools, с помощью которого вы имеете возможность изменить текст строки, ее центрирование и цвет, как и раньше. Если вы щелкнете на пиктограмме Back панели инструментов Explorer, будет перезагружен предыдущий открытый вами документ. Если вы перед щелчком на Back измените документ ShowString, вам даже будет предложено сохранить измененный документ! Microsoft планирует в следующем поколении Windows интегрировать рабочий стол с помощью интерфейса, подобного интерфейсу Internet Explorer. То, что вы сейчас видите, является эскизом будущего продукта.

При помощи соответствующей настройки мастера AppWizard приложению можно придать и функциональные возможности контейнера документов ActiveX. Возможно, вы уже обратили внимание на соответствующий флажок в диалоговом окне Step 3 of 6. Настроить подобным образом приложение не сложнее, чем настроить его на выполнение функций сервера документов ActiveX. Если вы хотите обеспечить пользователям своих приложений возможность "не отходя от кассы" открывать документы Word, Excel и им подобные, подумайте о такой настройке.

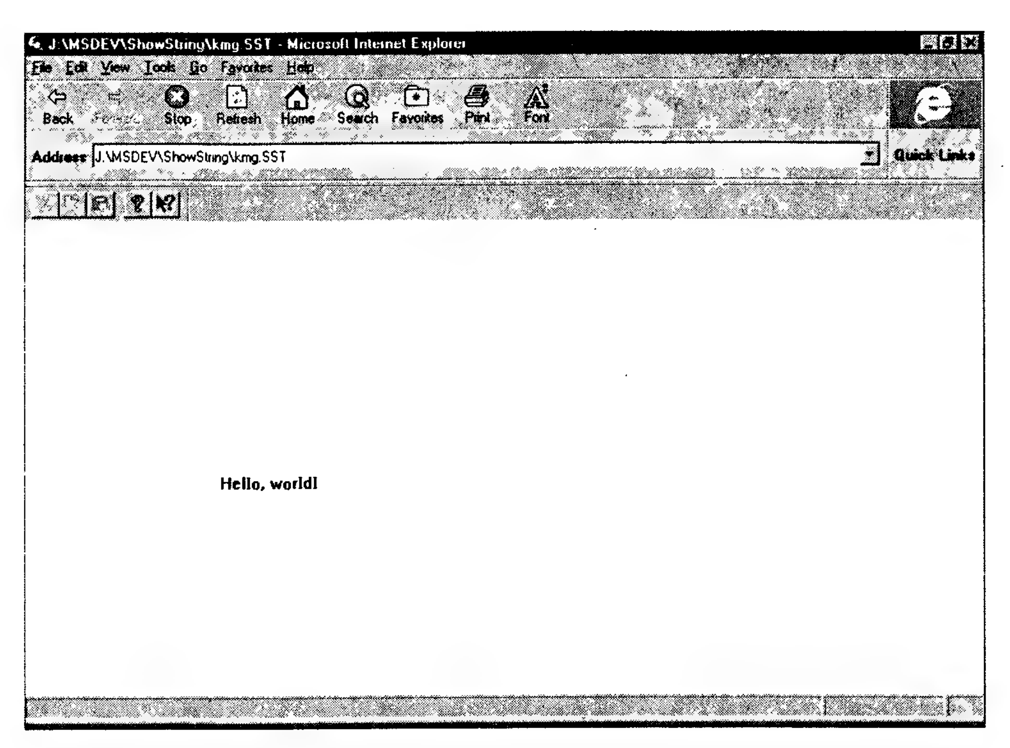


Рис. 15.18. Теперь Internet Explorer приобрел способность читать и записывать файлы приложения `ShowString`

ГЛАВА 16

Создание сервера автоматизации

В этой главе...

Снова проектируем ShowString

Создание приложения-контроллера в Visual Basic

Типы библиотек и внутренние механизмы ActiveX

Снова проектируем ShowString

Автоматизация, прежде носившая название *автоматизация OLE* (OLE Automation), а теперь именуемая *автоматизация ActiveX* (ActiveX Automation), подразумевает технологию создания программ, элементы и составные части которых могут вызываться из других программ. Прочие программы смогут вызывать созданную вами программу не в виде подключаемого модуля DLL, а непосредственно. Программисты говорят, что такая программа *предоставляет* (expose) как методы (функции), так и свойства (переменные) для использования в других приложениях. Достоинством технологии автоматизации ActiveX является то, что для приложения сервера автоматизации нет необходимости создавать специальный макроязык. Достаточно просто поместить в приложение определенные "зацепки", предназначенные для более универсального макроязыка (*Visual Basic for Application*), которые позволят ему связываться с элементами приложения.

Если вы точно выполняли все примеры в предыдущих главах этой книги, то, вероятно, уже можете сформировать заготовку приложения ShowString с закрытыми глазами. Тем не менее я предлагаю создать ShowString еще раз. В данном варианте приложению не потребуется иметь меню Tools, а соответственно и включенную в него команду Options. Вместо этого другие программы будут непосредственно выполнять ввод текста строки и управлять режимами ее отображения. Переменные-члены в классе документа в сравнении с другими версиями ShowString останутся прежними, не изменится и текст функции OnDraw().

Заготовка сервера автоматизации, созданная AppWizard

Для построения очередной версии ShowString, являющейся сервером автоматизации, прежде всего воспользуйтесь AppWizard, чтобы создать заготовку программы. Запустите AppWizard, как обычно, но укажите каталог, отличный от того, в котором создавались прежние версии ShowString. При настройке мастера выполните практически те же установки опций, что и прежде: имя приложения ShowString, приложение с поддержкой многодокументного интерфейса. Однако поддержка баз данных отсутствует. На этапе 3 настройки AppWizard в группе What compound document support would you like to include (Какую поддержку составных документов будете включать в приложение?) установите переключатель None (самый верхний переключатель в диалоговом окне), но дополнительно установите флажок Automation опции поддержки автоматизации. В последующих шагах мастера AppWizard выберите поддержку стационарной панели инструментов, строки состояния, печати и режима предварительного просмотра распечатки документа, контекстно-зависимой справки и объемного дизайна элементов управления. На последней странице потребуйте создания комментариев в текстах программ и использования разделяемых модулей DLL.

На заметку Несмотря на то что рассматриваемая технология теперь называется ActiveX и термин "автоматизация ActiveX" все чаще заменяется просто словом "автоматизация", в диалоговых окнах мастера AppWizard используются ссылки на технологию составных документов (Compound Document). Кроме того, во многих именах классов, обсуждаемых в этой главе, содержится аббревиатура OLE, а в комментариях присутствуют ссылки на технологию OLE. Хотя Microsoft уже изменила название этой технологии, соответствующие изменения в Visual C++ еще не проведены. Сейчас нам не остается ничего другого, как смириться с указанными противоречиями вплоть до выхода следующей версии Visual C++.

Во вновь созданном приложении имеется лишь несколько отличий от заготовки приложения без поддержки автоматизации. В основном они сосредоточены в классах приложения и документа.

Класс CShowStringApp

В класс приложения CShowStringApp внесено несколько изменений. В исходный файл непосредственно перед функцией InitInstance() вставлен текст, приведенный в листинге 16.1.

Листинг 16.1. Файл ShowString.cpp — определение CLSID

```
// Этот идентификатор был специально сгенерирован для вашего
// приложения с тем, чтобы обеспечить его статистическую
// уникальность. Можете его изменить, если предпочитаете
// присвоить вашему приложению конкретное значение идентификатора.

// {61C76C05-70EA-11D0-9AFF-0080C81A397C}
static const CLSID clsid =
{ 0x61c76c05, 0x70ea, 0x11d0, { 0x9a, 0xff, 0x0, 0x80, 0xc8,
   0x1a, 0x39, 0x7c } };
```

В программе, сгенерированной на вашем компьютере, эти числа будут иметь другие значения. Данный идентификатор класса (Class ID — CLSID) уникально определяет ваше приложение-сервер автоматизации.

В функцию CShowStringApp::InitInstance() внесено несколько изменений. Фрагмент программы, текст которого приведен в листинге 16.2, инициализирует библиотеки ActiveX (OLE).

Листинг 16.2. Файл ShowString.cpp — инициализация библиотек

```
// Инициализация библиотек OLE.
if (!AfxOleInit())
{
   AfxMessageBox(IDP_OLE_INIT_FAILED);
   return FALSE;
}
```

Как и в случае приложения-сервера ActiveX, в функции InitInstance() после выполнения инициализации шаблона документа осуществляется подключение шаблона документа к объекту класса COleTemplateServer:

```
m_server.ConnectTemplate(clsid, pDocTemplate, FALSE);
```

Затем функция InitInstance() проверяет, не был ли сервер запущен для редактирования внедренного объекта или в качестве сервера автоматизации. Если это так, то нет необходимости выводить на экран главное окно, поэтому сразу же осуществляется выход из функции, как показано в листинге 16.3.

Листинг 16.3. Файл ShowString.cpp — анализ типа запуска приложения

```
// Приложение запущено как сервер OLE?
if (cmdInfo.m_bRunEmbedded || cmdInfo.m_bRunAutomated)
{

   // Приложение было запущено с ключом /Embedding или
```

```
    // /Automation. В данном случае нет необходимости отображать
    // на экране главное окно приложения.
    return TRUE;
}

// Когда серверное приложение запускается как отдельное
// самостоятельное задание, полезно будет обновить системный
// реестр на тот случай, если он был поврежден.
m_server.UpdateRegistry(OAT_DISPATCH_OBJECT);
COleObjectFactory::UpdateRegistryAll();
```

Если ShowString было запущено как отдельное самостоятельное приложение, фрагмент программы, текст которого представлен в листинге 16.3, осуществляет обновление системного реестра, о чем шла речь в главе 15.

Класс CShowStringDoc

Класс документа CShowStringDoc, как и прежде, наследует не от класса документа OLE, а от класса CDocument, но на этом сходство с прежним не-OLE-классом CShowStringDoc и заканчивается. Первый фрагмент вновь добавленного текста программы, представленный в листинге 16.4, находится в файле ShowStringDoc.cpp сразу после карты сообщений.

Листинг 16.4. Файл ShowStringDoc.cpp — карта диспетчера

```
BEGIN_DISPATCH_MAP(CShowStringDoc, CDocument)
    //{{AFX_DISPATCH_MAP(CShowStringDoc)

        // ВНИМАНИЕ: ClassWizard будет здесь размещать и удалять
        // макросы переназначения.

        // НЕ РЕДАКТИРУЙТЕ то, что находится в данном
        // блоке.
    //}}AFX_DISPATCH_MAP
END_DISPATCH_MAP()
```

На текущий момент *карта диспетчера* пуста. Карта диспетчера подобна карте сообщений в том, что с ее помощью выполняется подключение событий окружающей среды к функциям, определенным в классах C++. Когда вы будете определять с помощью ClassWizard методы и свойства данного документа, предоставляемые другим приложениям, карта диспетчера будет обновляться.

После карты диспетчера добавлен еще один уникальный идентификатор, IID (Interface identifier — идентификатор интерфейса). Как следует из листинга 16.5, IID так же, как и CLSID, определяется как статическая переменная-член.

Листинг 16.5. Файл ShowStringDoc.cpp — идентификатор интерфейса IID

```
// Внимание: в программу добавляется переменная IID_IShowString,
// предназначенная для обеспечения связывания программы с VBA.
// Данный IID должен соответствовать GUID, который присвоен
// диспинтерфейсу в файле .ODL.

// {61C76C07-70EA-11D0-9AFF-0080C81A397C}
static const IID IID_IShowString =
{ 0x61c76c07, 0x70ea, 0x11d0, { 0x9a, 0xff, 0x0, 0x80,
    0xc8, 0x1a, 0x39, 0x7c } };
```

Далее размещена *карта интерфейсов*, которая выглядит следующим образом:

```
BEGIN_INTERFACE_MAP(CShowStringDoc, CDocument)
   INTERFACE_PART(CShowStringDoc, IID_IShowSt, Dispatch)
END_INTERFACE_MAP()
```

Карта интерфейсов скрывает функции ActiveX (такие, как QueryInterface()) от вас, программиста, и, как и карта сообщений, обеспечивает возможность работать на более абстрактном уровне. Приложению ShowString не потребуется большого количества элементов в карте интерфейсов, но для многих других приложений количество элементов может быть значительно больше. ClassWizard берет на себя установку элементов в карту интерфейсов.

Конструктор документа также претерпел некоторые изменения. Текст программы, сгенерированный мастером AppWizard, представлен в листинге 16.6.

Листинг 16.6. Файл ShowStringDoc.cpp — конструктор класса

```
CShowStringDoc::CShowStringDoc()
{

   // TODO: поместите сюда специфический для вашего
   // приложения текст, который должен выполняться в конструкторе.
   EnableAutomation();
   AfxOleLockApp();
}
```

Назначение функции EnableAutomation() соответствует ее имени — она задействует механизм автоматизации для данного документа. Функция AfxOleLockApp() используется для того, чтобы не позволить приложению закончить работу, если хотя бы один его документ все еще где-то используется. Представим себе ситуацию, в которой пользователь открывает одновременно два приложения, применяющие объекты ShowString. Когда первое приложение будет закрыто, ShowString должно продолжать работу, поскольку оно должно обслуживать второе приложение. В технологии ActiveX эта проблема разрешается методом создания в приложении счетчика, хранящего число активных объектов, использующих данное приложение. Функция AfxOleLockApp() увеличивает значение этого счетчика на единицу. Если в момент, когда пользователь делает попытку закрыть приложение, этот счетчик имеет отличное от нуля значение, данное приложение будет убрано с экрана, но не закрыто.

Теперь будет понятен и текст деструктора класса документов ShowString:

```
CShowStringDoc::~CShowStringDoc()
{
   AfxOleUnlockApp();
}
```

Функция AfxOleUnlockApp() уменьшает на единицу значение счетчика активных объектов приложения, обеспечивая таким образом возможность закрытия сервера автоматизации в тот самый момент, когда отпадает необходимость в его услугах.

Предоставляемые свойства

На данный момент мы имеем сервер автоматизации, который не предоставляет внешним программам ни методов, ни свойств. Кроме того, четыре переменные-члены класса документа, использовавшиеся во всех предыдущих версиях ShowString, все еще отсутствуют в новой версии приложения. Вот эти четыре переменные-члены:

- string — отображаемая на экране строка;

- color — цвет текста строки; значения: 0 — для черного, 1 — для красного, 2 — для зеленого;
- horizcenter — содержит значение TRUE, если строка должна центрироваться по горизонтали;
- vertcenter — содержит значение TRUE, если строка должна центрироваться по вертикали.

Данные переменные будут добавлены в приложение как свойства автоматизации, так что не следует отдельно вводить их имена в определение класса CShowStringDoc. Раскройте на экране окно ClassWizard, щелкнув на его пиктограмме на панели инструментов или выбрав команду View⇒ClassWizard. Для включения в приложение свойств и методов используется вкладка Automation, показанная на рис. 16.1. Щелкните на ее корешке.

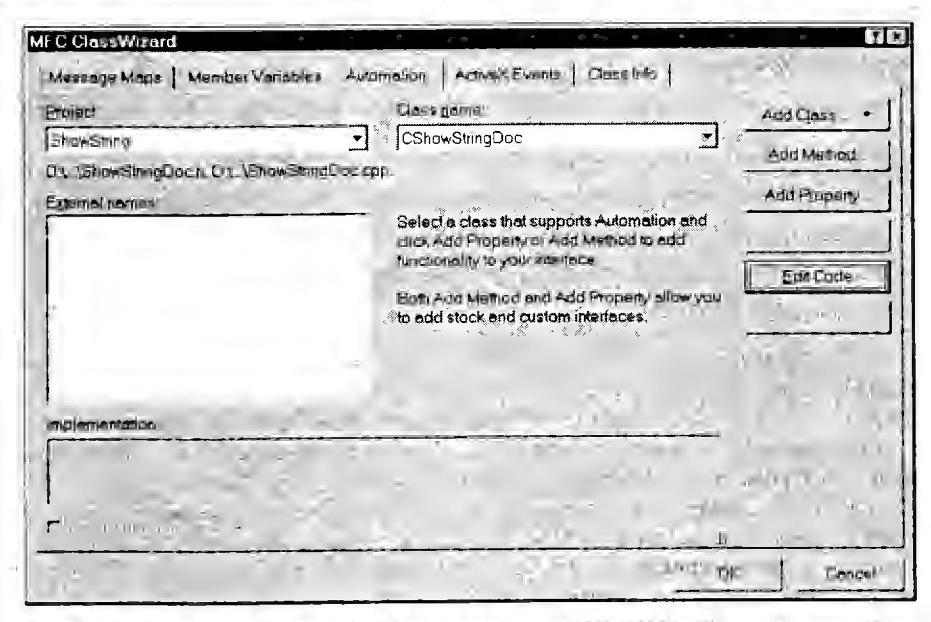


Рис. 16.1. На вкладке Automation диалогового окна ClassWizard выполняется большая часть работы по созданию сервера автоматизации

Первым шагом на пути восстановления функциональных возможностей прежнего ShowString является добавление в класс документа переменных-членов, которые будут предоставляться как свойства сервера автоматизации. Существует два варианта предоставления свойств: как переменных и посредством специальных функций. Первый вариант заключается просто в объявлении открытых переменных-членов класса C++, после чего другие приложения смогут непосредственно считывать или изменять значения этих переменных. При поступлении извне требования получить доступ к переменной вызываются соответствующими извещающими функциями сервера. Второй вариант — предоставление свойства через функции Get и Set — состоит в определении закрытых переменных-членов, имеющих открытые методы доступа к ним. В прочих приложениях осуществляется прямой доступ к этим переменным, но управляющая система преобразует обращение к свойствам в вызовы соответствующих функций Get и Set. При разработке функции Get можно предусмотреть контроль состояния объекта (например, проверить, чтобы отсортированный список действительно был отсорти-

рован в данный момент или чтобы итог был вычислен), прежде чем возвращать значение свойства. В свою очередь, при разработке функции Set можно предусмотреть контроль допустимости передаваемого значения или же вычисление значения других переменных, связанных с изменяемым свойством. Чтобы обеспечить для свойства режим доступа только для чтения, можно объявить его как имеющее функции Get и Set, а затем реализовать только функцию Get.

В нашем примере добавим в класс CShowStringDoc два свойства, определяющих центрирование строки, как имеющие функции Get и Set. Саму же строку и свойство, определяющее ее цвет, оформим как свойства с непосредственным доступом. Для этого выполните следующие операции.

1. Убедитесь, что выбранным является класс CShowStringDoc, а затем щелкните на кнопке Add Property и тем самым откройте одноименное диалоговое окно.
2. В поле ввода External name введите значение String, а ClassWizard выполнит остальную работу, заполнив значениями поля Variable name (Имя переменной) и Notification function (Функция уведомления).
3. В раскрывающемся списке Type (Тип) выберите значение CString, после чего диалоговое окно должно приобрести вид, показанный на рис. 16.2.

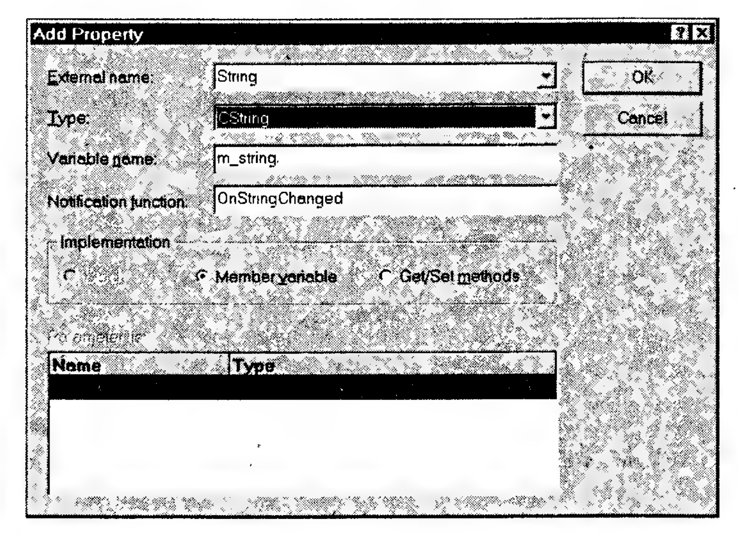


Рис. 16.2. Добавление String в качестве свойства с прямым доступом

4. Щелкните на кнопке OK. Вновь щелкните на кнопке Add Property и добавьте переменную Color, как переменную с прямым доступом (рис. 16.3). Установите для нее тип данных short.
5. Щелкните на кнопке OK. Вновь щелкните на кнопке Add Property и добавьте переменную HorizCenter.
6. Определите для нее тип BOOL, а затем в группе Implementation установите переключатель Get/Set methods. Имена полей Variable name и Notification function изменятся на Get function и Set function, причем эти поля будут автоматически заполнены значениями, как показано на рис. 16.4. (Если при этом тип переменной будет изменен, вновь установите для нее тип BOOL.) Щелкните на кнопке OK.
7. Добавьте переменную VertCenter точно так, как это было сделано для переменной HorizCenter.

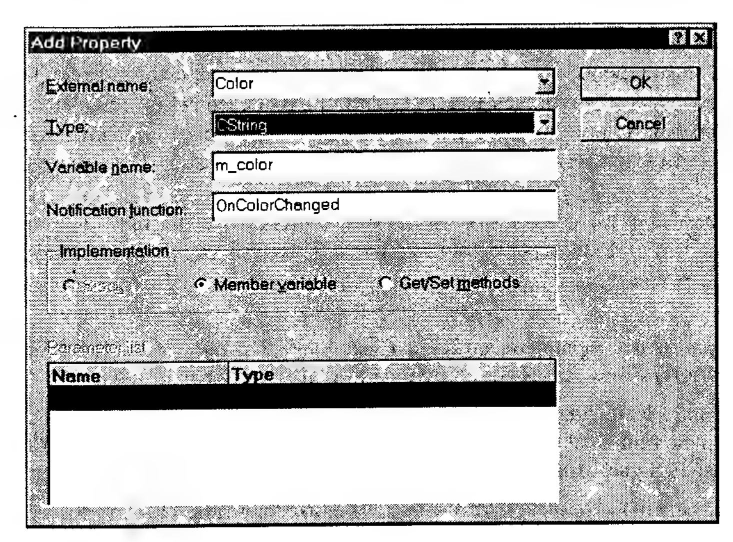


Рис. 16.3. Добавление `Color` *в качестве свойства с прямым доступом*

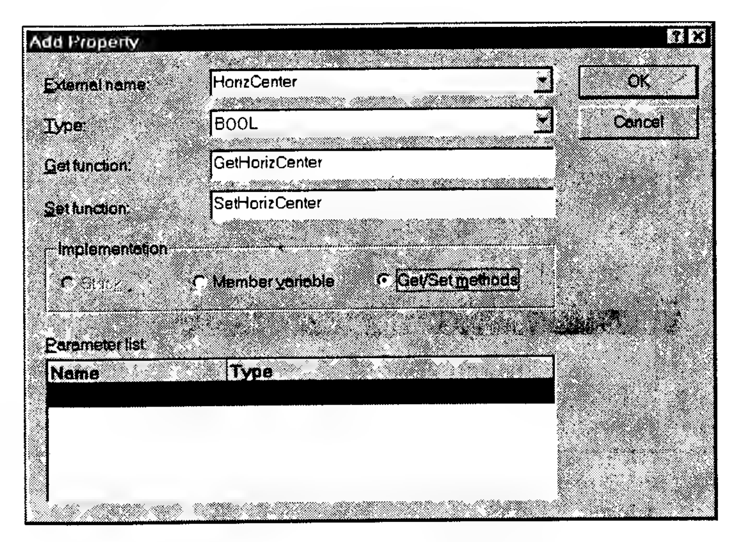


Рис. 16.4. Добавление `HorizCenter` *в качестве свойства с методами* `Get/Set`

Внимание!

Как только вы сделаете щелчок на кнопке OK, изменить тип, внешнее имя и прочие параметры создаваемого свойства будет невозможно. В случае ошибки необходимо удалить это свойство, а потом вновь добавить его с новым именем, типом и т.д. Прежде чем щелкнуть на кнопке OK, всегда внимательно анализируйте содержимое диалогового окна Add Property.

На рис. 16.5 показана отображаемая ClassWizard сводная информация о предоставляемых свойствах и методах. Детальные сведения по каждому свойству выводятся в поле Implementation, которое расположено под списком внешних имен свойств. На рис. 16.5 выбрано свойство VertCenter, и в окне Implementation сообщается, что данное свойство имеет функции Get и Set, а также приводятся их прототипы. Для закрытия окна ClassWizard щелкните на кнопке OK.

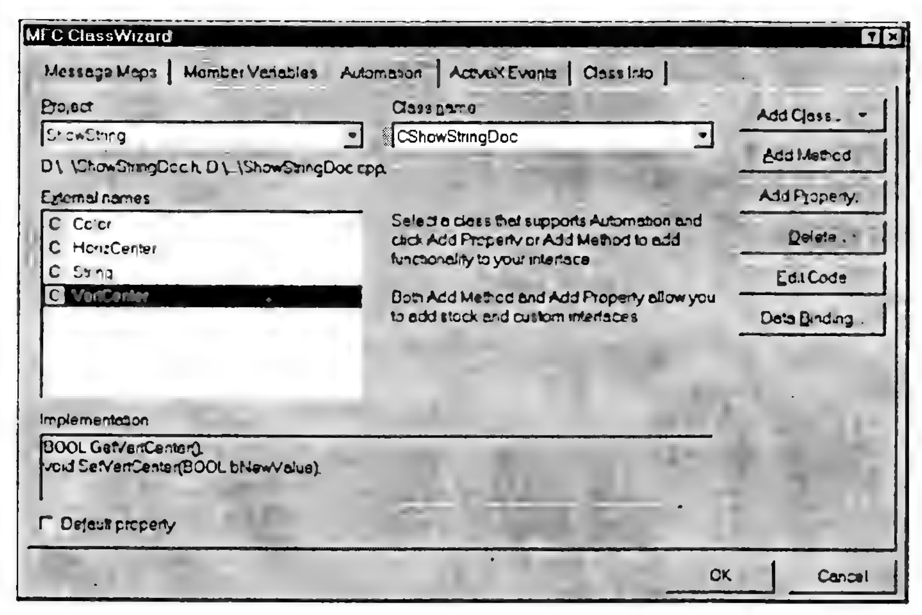


Рис. 16.5. ClassWizard предоставляет сводные данные по добавленным свойствам

Нет ничего удивительного в том, что в результате проведенных добавлений ClassWizard внес изменения в файлы заголовка и исходного текста для класса CShowStringDoc. Новая карта диспетчера в файле заголовка имеет вид, представленный в листинге 16.7.

Листинг 16.7. Файл ShowStringDoc.h — карта диспетчера

```
//{{AFX_DISPATCH(CShowStringDoc)
CString m_string;
afx_msg void OnStringChanged();
short m_color;
afx_msg void OnColorChanged();
afx_msg BOOL GetHorizCenter();
afx_msg void SetHorizCenter(BOOL bNewValue);
afx_msg BOOL GetVertCenter();
afx_msg void SetVertCenter(BOOL bNewValue);
//}}AFX_DISPATCH
DECLARE_DISPATCH_MAP()
```

Были добавлены две новые переменные-члены: m_string и m_color.

На заметку

Естественно, возникает вопрос, действительно ли эти переменные объявлены как открытые переменные-члены? Нет, это не так. Непосредственно перед картой диспетчера имеется следующий макрос:

```
DECLARE_MESSAGE_MAP
```

Если раскрыть его, то можно увидеть объявление нескольких защищенных переменных. Поскольку интересующие нас переменные и функции объявляются сразу после переменных, о которых сказано выше, они также являются защищенными переменными и защищенными функциями. Доступ к ним выполняется таким же образом, как и к защищенным функциям, перехватывающим сообщения. Они вызываются спрятанными в классе функциями-членами, которые и координируют движение сообщений на основе соответствующих карт.

В исходный файл был добавлен фрагмент программного текста, однако он не отличается глубиной мысли, что видно из листинга 16.8.

Листинг 16.8. Файл ShowStringDoc.cpp — функции уведомления Get и Set

```
/////////////////////////////////////////////////////////////

// Команды класса CShowStringDoc.

void CShowStringDoc::OnColorChanged()
{

   // TODO: вставьте подпрограмму обработки уведомления.
}

void CShowStringDoc::OnStringChanged()
{

   // TODO: вставьте подпрограмму обработки уведомления.
}

BOOL CShowStringDoc::GetHorizCenter()
{

   // TODO: вставьте подпрограмму обработки свойства.
   return TRUE;
}

void CShowStringDoc::SetHorizCenter(BOOL bNewValue)
{

   // TODO: вставьте подпрограмму обработки свойства.
}

BOOL CShowStringDoc::GetVertCenter()
{

   // TODO: вставьте подпрограмму обработки свойства.
   return TRUE;
}

void CShowStringDoc::SetVertCenter(BOOL bNewValue)
{

   // TODO: вставьте подпрограмму обработки свойства.
}
```

Класс документа все еще не имеет переменных-членов, содержащих флажки центрирования. Добавьте их в файл заголовка в качестве закрытых переменных-членов.

```
// Attributes
private:
    BOOL m_horizcenter;
    BOOL m_vertcenter;
```

А теперь нужно написать для этих переменных функции Set и Get. Их тексты приведены в листинге 16.9.

Листинг 16.9. Файл ShowStringDoc.cpp — функции Get и Set для переменных флажков центрирования

```
BOOL CShowStringDoc::GetHorizCenter()
{
    return m_horizcenter;
}

void CShowStringDoc::SetHorizCenter(BOOL bNewValue)
{
    m_horizcenter = bNewValue;
}

BOOL CShowStringDoc::GetVertCenter()
{
    return m_vertcenter;
}

void CShowStringDoc::SetVertCenter(BOOL bNewValue)
{
    m_vertcenter = bNewValue;
}
```

Функция OnDraw()

Определив переменные-члены, мы наполовину завершили работу по восстановлению функциональных возможностей ShowString. После внесения изменений в функцию OnDraw() класса представления будет выполнена и почти вся оставшаяся часть работы.

Чтобы написать новую версию функции OnDraw(), которая будет правильно отображать строку, откройте старую версию ShowString, созданную в главе 8. Вставьте в новую функцию указанные ниже фрагменты текста. (Если что-то в этих текстах будет вам непонятно, найдите в главе 8 их исчерпывающее объяснение.) Прежде всего в функции CShowStringDoc::OnNewDocument() (листинг 16.10) следует инициализировать переменные-члены.

Листинг 16.10. Файл ShowStringDoc.cpp — функция CShowStringDoc::OnNewDocument()

```
BOOL CShowStringDoc::OnNewDocument()
{
    if (!CDocument::OnNewDocument())
        return FALSE;

    m_string = "Hello, world!";
    m_color = 0; //Черный.
    m_horizcenter = TRUE;
    m_vertcenter = TRUE;
```

```
    return TRUE;
}
```

Далее отредактируйте функцию класса документа `Serialize()`. Ее новый текст приведен в листинге 16.11.

Листинг 16.11. Файл `ShowStringDoc.cpp` **— функция** `CShowStringDoc::Serialize()`

```
void CShowStringDoc::Serialize(CArchive& ar)
{
    if (ar.IsStoring())
    {
        ar << m_string;
        ar << m_color;
        ar << m_horizcenter;
        ar << m_vertcenter;
    }
    else
    {
        ar >> m_string;
        ar >> m_color;
        ar >> m_horizcenter;
        ar >> m_vertcenter;
    }
}
```

И наконец, откорректируйте функцию класса представления `OnDraw()` так, чтобы она действительно выводила строку на экран (листинг 16.12).

Листинг 16.12. Файл `ShowStringView.cpp` **— функция** `CShowStringView::OnDraw()`

```
void CShowStringView::OnDraw(CDC* pDC)
{
    CShowStringDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    COLORREF oldcolor;
    switch (pDoc->GetColor())
    {
    case 0:
        oldcolor = pDC->SetTextColor(RGB(0,0,0)); //Черный.
        break;
    case 1:
        oldcolor = pDC->SetTextColor(RGB(0xFF,0,0)); //Красный.
        break;
    case 2:
        oldcolor = pDC->SetTextColor(RGB(0,0xFF,0)); //Зеленый.
        break;
    }

    int DTflags = 0;
    if (pDoc->GetHorizcenter())
    {
        DTflags |= DT_CENTER;
    }
    if (pDoc->GetVertcenter())
    {
        DTflags |= (DT_VCENTER|DT_SINGLELINE);
    }
```

```
    CRect rect;
    GetClientRect(&rect);
    pDC->DrawText(pDoc->GetString(), &rect, DTflags);
    pDC->SetTextColor(oldcolor);

}
```

Когда с помощью ClassWizard вы добавляли переменные m_string, m_color, m_horizcenter и m_vertcenter в класс документа, они были добавлены в него как защищенные переменные-члены. В только что введенном фрагменте программы необходимо было иметь к ним доступ. Как видите, представление с этой целью вызывает открытые функции доступа к ним.

На заметку

Вместо использования функций доступа вы могли бы выбрать вариант объявления класса представления дружественным для класса документа, чтобы получить прямой доступ к требуемым переменным-членам. Однако в этом случае функции класса представления получили бы еозможность использовать и изменять *все* защищенные и закрытые переменные-члены класса документа. Выбранный нами вариант ограничивает доступ лишь в пределах необходимого и не нарушает инкапсуляции.

В классе документа уже существует несколько функций, которые имеют доступ к данным переменным, но они являются защищенными функциями и используются подпрограммами ActiveX. Четыре открытые функции, которые вы должны добавить, имеют не очень удачные имена. Поместите их в файл ShowStringDoc.h так, как показано в листинге 16.13.

Листинг 16.13. Файл ShowStringDoc.h — функции открытого доступа

```
public:
    CString GetDocString() {return m_string;}
    int GetDocColor() {return m_color;}
    BOOL GetHorizcenter() {return m_horizcenter;}
    BOOL GetVertcenter() {return m_vertcenter;}
```

В функции CShowStringView::OnDraw() замените вызов функции GetColor() вызовом GetDocColor(), а вызов функции GetString() — вызовом GetDocString(). Оттранслируйте проект и убедитесь в отсутствии мелких ошибок или пропуска изменеиий. Вероятио, у вас возникнет желание немедленно запустить на выполнение новое приложение ShowString. Не следует этого делать, поскольку приложение не будет работать до тех пор, пока мы не внесем еще несколько изменений.

Вывод окна на экран

По умолчанию сервер автоматизации не имеет главного окна. Вернемся к функции CShowStringApp::InitInstance(), небольшой фрагмент текста которой представлен в листинге 16.14.

Листинг 16.14. Файл ShowString.cpp — режим запуска приложения

```
// Контроль: приложение запущено как сервер OLE?
if (cmdInfo.m_bRunEmbedded || cmdInfo.m_bRunAutomated)
{
    // Приложение было запущено с параметром /Embedding или
    // /Automation. В данном случае нет необходимости отображать
    // на экране главное окно приложения.
    return TRUE;
}
```

В данном фрагменте программы управление возвращается *до* вывода на экран главного окна. Если удалить этот фрагмент программы, ShowString будет всегда выводить на экран свое главное окно. Однако мы воспользуемся более общим способом и добавим метод ShowWindow(), предназначенный для вызова из приложения-контроллера. Кроме того, потребуется добавить метод RefreshWindow(), который будет обновлять представление после изменения переменных. Добавление этих функций с помощью ClassWizard не составляет большого труда. Откройте окно ClassWizard, выберите вкладку Automation, убедитесь, что выбран именно класс CShowStringDoc, а затем щелкните на кнопке Add Method. В поле ввода External Name введите значение ShowWindow. Значение в поле Internal name будет автоматически помещено в ClassWizard — изменять его нет необходимости. В раскрывающемся списке Return type выберите значение void. После заполнения полей окно должно иметь вид, показанный на рис. 16.6.

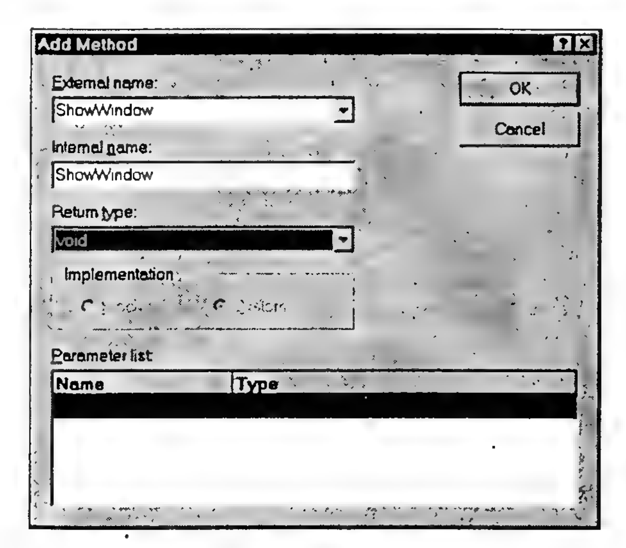


Рис. 16.6. Добавление метода ShowWindow() очень просто выполняется с помощью ClassWizard

Щелкните на кнопке OK, и функция ShowWindow() появится в середине списка свойств. Оказывается, этот список предназначен для вывода в алфавитном порядке перечня свойств и методов. Символ C перед именем свойства напоминает о том, что данное свойство является пользовательским. Символ M перед именем метода указывает на то, что это именно имя метода. Выделив в списке значение ShowWindow(), щелкните на кнопке Edit Code, а затем введите текст этой функции, представленный в листинге 16.15.

Листинг 16.15. Файл ShowStringDoc.cpp — функция CShowStringDoc::ShowWindow()

```
void CShowStringDoc::ShowWindow()
{
   POSITION pos = GetFirstViewPosition();
   CView* pView = GetNextView(pos);
   if (pView != NULL)
   {
      CFrameWnd* pFrameWnd = pView->GetParentFrame();
      pFrameWnd->ActivateFrame(SW_SHOW);
      pFrameWnd = pFrameWnd->GetParentFrame();
```

```
        if (pFrameWnd != NULL)
            pFrameWnd->ActivateFrame(SW_SHOW);
    }
}
```

В данном фрагменте программы активизируется представление и запрашивается его вывод на экран. Снова выведите на экран окно ClassWizard и щелкните на кнопке Add Method, а затем добавьте метод RefreshWindow(), возвращающий значение void. Щелкните на кнопке OK, а затем на кнопке Edit Code. Текст функции RefreshWindow(), представленный в листинге 16.16, проще текста предыдущей функции.

Листинг 16.16. Файл ShowStringDoc.cpp — функция CShowStringDoc::RefreshWindow()

```
void CShowStringDoc::RefreshWindow()
{
    UpdateAllViews(NULL);
    SetModifiedFlag();
}
```

Эти операторы обеспечивают перерисовку представления (которое уже активно) и его родительского окна. И, поскольку изменения в документе практически всегда требуют перерисовки, это самое подходящее место для вызова функции SetModifiedFlag(), хотя можно поместить ее вызов в каждую из функций Set и функций уведомления (On...Changed()) для свойств прямого доступа. Теперь необходимо добавить вызов функции RefreshWindow() в каждую из только что упомянутых функций. В качестве примера в листинге 16.17 приведен текст функции SetHorizCenter().

Листинг 16.17. Файл ShowStringDoc.cpp — функция CShowStringDoc::SetHorizCenter()

```
void CShowStringDoc::SetHorizCenter(BOOL bNewValue)
{
    m_horizcenter = bNewValue;
    RefreshWindow();
}
```

Функция OnColorChange() будет иметь такой вид:

```
void CShowStringDoc::OnColorChanged()
{
    RefreshWindow();
}
```

Такой же вызов функции RefreshWindow() поместите в SetVertCenter() и OnStringChange(). Вот теперь все готово для трансляции и тестирования приложения. Оттранслируйте проект и исправьте возможные мелкие ошибки и упущения. Запустите ShowString как самостоятельное отдельное приложение с целью его регистрации и проверки функционирования его подпрограмм построения изображения. У вас нет возможности изменить строку, ее цвет или центрирование, как в предыдущих версиях ShowString, поскольку вновь созданная версия не поддерживает команду Tools⇨Options и ее диалоговое окно. В данной версии ShowString эта обязанность возлагается на приложение-контроллер автоматизации.

Создание приложения-контроллера в Visual Basic

В этой главе уже несколько раз упоминалось приложение-контроллер автоматизации и, вероятно, вам весьма интересно узнать, что это такое. Создать его предстоит вам, воспользовавшись для этого таким мощным инструментом, как Visual Basic. На рис. 16.7 показан интерфейс Visual Basic.



Если у вас нет Visual Basic, но имеется полный комплект файлов продукта Visual C++, можете использовать приложение DispTest, являющееся упрощенной версией Visual Basic и поставляемое в составе Visual C++. Оно не может быть добавлено в меню Пуск Windows 95, но вы всегда имеете возможность запустить программу DISPTEST.EXE, файл которой хранится в папке C:\MSDEV\BIN или на компакт-диске Visual C++ в папке \MSDEV\BIN. Если вам приходилось писать макросы VBA в Excel и имеется копия Excel, можете использовать и его. Для тестирования работы сервера автоматизации OLE не имеет значения, какое именно из перечисленных выше приложений вы выберете.

Построение приложения-контроллера для сервера автоматизации ShowString мы начнем с запуска Visual Basic. Затем создайте новый проект, выбрав для этого File⇨New и дважды щелкнув на Standard EXE. В окне справа вверху с именем Project1 щелкните на кнопке View Code. В раскрывшемся окне выберите в левом разворачивающемся списке значение Form, после чего на экране будет выведен текст подпрограммы Form_Load(). Введите в эту подпрограмму текст, представленный в листинге 16.18.

Листинг 16.18. Файл Form1.frm — язык программирования Visual Basic

```
Private Sub Form_Load ()
   Set ShowTest = CreateObject("ShowString.Document")
   ShowTest.ShowWindow
   ShowTest.HorizCenter = False
   ShowTest.Color = 1
   ShowTest.String = "Hello from VB"
   Set ShowTest = Nothing
End Sub
```

В левом раскрывающемся списке выберите значение General (общие), а затем введите такую строку:

```
Dim ShowTest As Object
```

Для тех, кто не знаком с языком Visual Basic, программы на нем станут понятнее, если выполнять их в пошаговом режиме. Для выполнения первой строки программы выберите команду Debug⇨Step Into. Затем для выполнения каждой очередной строки программы нажимайте клавишу <F8>. (После каждого нажатия подождите, пока курсор примет свою нормальную форму.) Строка в секции общих объявлений General определяет объект с именем ShowTest. Когда загружается форма (что происходит всякий раз, когда вы запускаете эту маленькую программу), создается экземпляр объекта ShowString. Следующая строка в процедуре Form_Load вызывает метод ShowWindow для отображения на экране главного окна. Как только отладчик делает очередную паузу, строка программы, которая будет выполняться следующей, подсвечивается.

Для выполнения строки, которая отключает горизонтальное центрирование, нажмите клавишу <F8>. Обратите внимание на то, что в программе не вызывается функция SetHorizCenter(). Созданный вами сервер автоматизации ActiveX предоставляет HorizCenter как свойство, и из программы на Visual Basic вы обращаетесь с ним именно так, как со свойством. Выполняющая система C++ для изменения свойства вызывает функцию SetHorizCenter() вместо того, чтобы просто выполнить изменение напрямую, а затем вызвать функцию уведомления для передачи сообщения о произведенных изменениях. После выполнения оператора установки нового значения изображение на экране должно измениться, поскольку метод SetHorizonCenter вызывает функцию RefreshWindow() для немедленной перерисовки экрана.

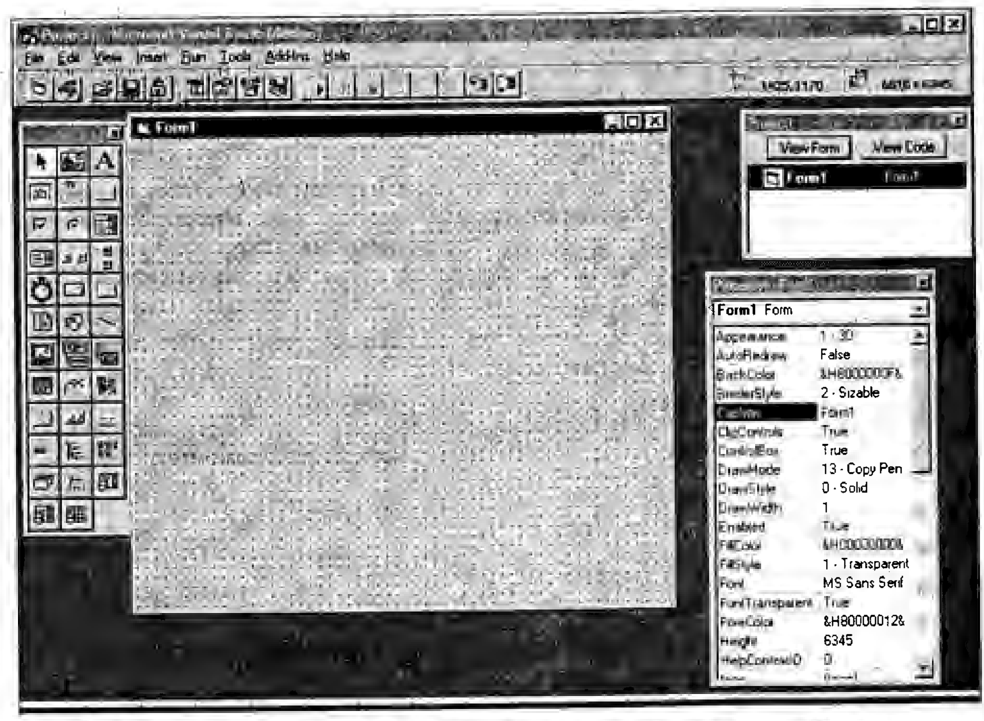


Рис. 16.7. С помощью Visual Basic приложение-контроллер создается очень быстро

Продолжайте пошаговое выполнение программы, нажимая клавишу <F8>, и строка в окне ShowString станет красной, а затем изменит свое значение на Hello from VB. Обратите внимание на то, что изменение этих непосредственно предоставляемых свойств внешне никак не отличается от изменения свойства HorizCenter, выполняемого посредством методов Set/Get. После завершения программы окно ShowString исчезнет. Вы успешно осуществили управление сервером автоматизации ActiveX из приложения Visual Basic.

Типы библиотек и внутренние механизмы ActiveX

Многих программистов пугает сложность ActiveX, и меньше всего они хотят знать, что скрыто за его таинственными покровами. И в этом нет ничего плохого. Действительно, естественное проявление объектно-ориентированного подхода — воспользоваться уже написанными средствами ActiveX для превращения оператора ShowTest.HorizCenter = False в вызов функции CShowStringDoc::SetHorizCenter(). Но чтобы узнать, как реализуется этот фокус или что делать в случае, если фокус не удался, необходимо добавить еще несколько элементов в мозаику общей картины происходящего. Вы уже видели карту диспетчера приложения ShowString, но вы еще не знакомы с библиотекой типов. Имеется в виду библиотека не в обычном смысле, а используемая программами ActiveX и системным реестром. Она создается как часть нормальной процедуры трансляции приложения из файла, содержащего описание данных на языке ODL (Object Definition Language — язык описания объектов). Этот файл был сгенерирован мастером AppWizard и при необходимости корректируется средствами ClassWizard.

Возможно, при выполнении трансляции приложения вы обратили внимание на новый элемент, появляющийся на вкладке ClassView. На рис. 16.8 этот элемент представлен в развернутом виде. Он содержит все свойства и методы, представляемые в интерфейсе IShowString созданного вами сервера автоматизации. Если в этом списке вы сделаете на элементе IShowString щелчок правой кнопкой мыши, раскроется контекстное меню, предоставляющее возможность добавить новые методы или свойства. Если вы сделаете двойной щелчок на любом из методов или свойств, будет открыт файл .ODL и соответствующий его фрагмент будет представлен на ваше обозрение. Текст файла ShowString.odl представлен в листинге 16.19.

Листинг 16.19. Файл ShowString.odl — библиотека типов приложения ShowString

```
// ShowString.odl : исходный текст библиотеки типов для ShowString.exe.

// Данный файл будет обрабатываться компилятором MIDL для
// создания библиотеки типов приложения (ShowString.tlb).

[ uuid(61C76C06-70EA-11D0-9AFF-0080C81A397C), version(1.0) ]
library ShowString
{
   importlib("stdole32.tlb");


   // Первичный диспинтерфейс для CShowStringDoc.

   [ uuid(61C76C07-70EA-11D0-9AFF-0080C81A397C) ]
   dispinterface IShowString
   {
      properties:

         // ВНИМАНИЕ: корректность информации о свойствах
         // поддерживается средствами ClassWizard; соблюдайте
         //предельную осторожность при редактировании данного текста.
         //{{AFX_ODL_PROP(CShowStringDoc)
         [id(1)] BSTR String;
         [id(2)] short Color;
         [id(3)] boolean HorizCenter;
         [id(4)] boolean VertCenter;
         //}}AFX_ODL_PROP
```

```
   methods:

      // ВНИМАНИЕ: корректность информации о методах
      // поддерживается средствами ClassWizard; соблюдайте
      //предельную осторожность при редактировании данного текста.
      //{{AFX_ODL_METHOD(CShowStringDoc)
      [id(5)] void ShowWindow();
      [id(6)] void RefreshWindow();
      //}}AFX_ODL_METHOD

  };


  // Информация о классе для CShowStringDoc.

  [ uuid(61C76C05-70EA-11D0-9AFF-0080C81A397C) ]
  coclass Document
  {
     [default] dispinterface IShowString;
  };

  //{{AFX_APPEND_ODL}}
  //}}AFX_APPEND_ODL}}
};
```

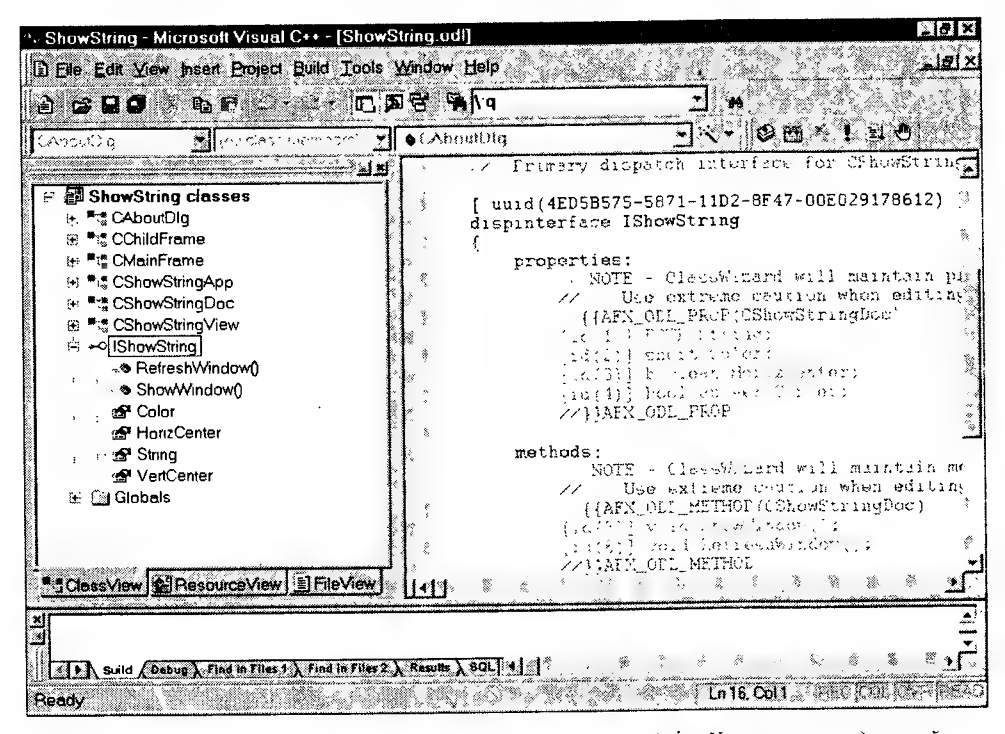


Рис. 16.8. На вкладке ClassView *сервера автоматизации ActiveX имеется отдельный элемент для каждого из его интерфейсов*

Теперь понятно, почему Visual Basic обрабатывал все четыре свойства, как обычные переменные: именно так они перечислены в файле ODL. Два предоставляемых метода тоже описаны здесь, в разделе методов. Причина, по которой вы передавали значение "ShowString.Document" функции CreateObject(), состоит в том, что в данном файле присутствует раздел coclass Document. Он указывает на диспетчерский интерфейс (диспинтерфейс), иосящий имя IShowString. Ниже приведена карта интерфейсов из файла ShowStringDoc.cpp.

```
BEGIN_INTERFACE_MAP(CShowStringDoc, CDocument)
   INTERFACE_PART(CShowStringDoc, IID_IShowString, Dispatch)
END_INTERFACE_MAP()
```

Поэтому вызов функции CreateObject("ShowString.Document") производит обращение к секции coclass файла ODL, в которой указывается интерфейс IShowString. Карта интерфейсов указывает от интерфейса IShowString на класс CShowStringDoc, имеющий карту диспетчера, подключающую свойства и методы внешней среды к коду C++. Можете мне поверить, что редактирование любой из этих секций вручную может иметь самые катастрофические последствия. Доверьте мастерам Visual C++ выполнять эту работу вместо вас.

В этой главе мы построили сервер автоматизации и организовали управление им из приложения, созданного в среде Visual Basic. Серверы автоматизации представляют собой намиого более мощное средство интеграции приложений по сравнеиию с существовавшими ранее. Но тот сервер, который был создан вами, не имеет средств взаимодействия с пользователем. Если в программе на Visual Basic необходимо предоставить пользователю возможность выбирать цвет, то ее реализация должна быть выполнена именно в программе на Visual Basic. Следующим логичным шагом на пути интеграции будет предоставление возможности небольшим внедренным объектам реагировать на действия, выполняемые пользователем, такие как щелчки и перетаскивания, а затем сообщать программе-контроллеру о том, что произошло. Именно это и выполняют элементы управления ActiveX, о которых пойдет речь в следующей главе.

ГЛАВА 17

Создание элемента управления ActiveX

В этой главе...

Создание элемента управления в виде игральной кости

Отображение текущего значения

Имитация броска кости в ответ на щелчок мышью

Совершенствование пользовательского интерфейса

Окна свойств

Имитация броска кости по требованию

Доработка программы

Элементы управления ActiveX призваны заменить собой элементы управления OLE, но различия между ними заключаются преимущественно в названии. (Большая часть документации, издаваемой Microsoft, по-прежнему ссылается на элементы управления OLE.) Уникальные возможности, которыми обладали эти, прежде именовавшиеся OLE, элементы управления, только расширились после перехода к ActiveX. Данная глава основывается на сведениях об ActiveX, которые вы получили из предыдущих глав. Элементы управления ActiveX очень похожи на серверы автоматизации ActiveX, но, в отличие от последних, элементы управления дополнительно обладают возможностью экспортировать *события*. А это позволяет непосредственно управлять поведением контейнера, который их содержит.

Элементы управления ActiveX пришли на смену элементам управления VBX. Последние широко использовались при создании 16-разрядных приложений для Windows, поскольку обеспечивали программистам возможность расширения стандартного набора элементов управления, предоставлявшегося компилятором. Первоначальной целью создания элементов управления VBX как раз и было предоставление программистам возможности использовать необычные элементы управления в пользовательском интерфейсе приложений. Создание элементов управления, имевших вид манометров или регуляторов громкости, уже не требовало значительных затрат времени и труда. Очень скоро программисты VBX перешли от создания простеньких элементов управления к разработке модулей, которые выполняли значительный объем вычислений или обработки данных. Точно так же многие из элементов управления ActiveX являются не просто элементами управления, а *компонентами*, которые легко могут быть использованы для быстрого создания мощных приложений.

Создание элемента управления в виде игральной кости

Приложением, которое мы будем рассматривать в качестве примера в данной главе, является имитатор бросания игрального кубика, одного из пары, образующей игральные кости. Представьте себе изображение игрального кубика со знакомым узором из точек, указывающих на текущее значение. Если пользователь делает щелчок на этом изображении, текущее значение изменяется на новое, случайным образом выбранное число, в диапазоне от 1 до 6. По окончании работы вы сможете встроить одну или более таких игральных костей в любую вашу программу.

Создание оболочки управляющего элемента

Процедура создания имитатора игральной кости начинается, как и всегда, с мастера AppWizard. Запустите Visual Studio, а затем выберите команду File⇨New. Щелкните на корешке вкладки Projects и выберите в списке, расположенном в левой части окна New, значение MFC ActiveX ControlWizard. В поле ввода Project name (справа вверху) введите имя проекта, укажите папку, в которую будут помещены файлы проекта, а затем щелкните на кнопке OK. На рис. 17.1 показано диалоговое окно проекта Dieroll с заполненными полями ввода.

На заметку Хотя в настоящее время обсуждаемая технология называется ActiveX, в составе многих имен классов, которые используются в этой главе, содержится аббревиатура OLE, а в комментариях присутствуют ссылки на OLE. Microsoft уже изменила название этой технологии, но соответствующие изменения в Visual C++ еще не проведены. Придется смириться с указанными противоречиями вплоть до выхода следующей версии Visual C++.

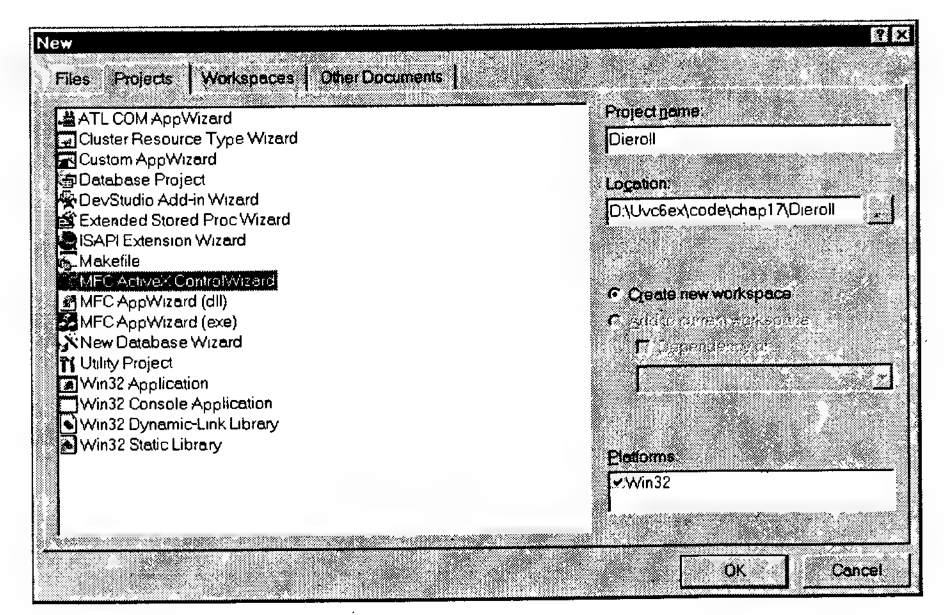


Рис. 17.1. Благодаря мастеру `AppWizard` *создание элемента управления ActiveX не требует больших усилий*

Настройка `AppWizard` для создания элемента управления ActiveX включает два этапа. Заполните поля в окне этапа 1 так, как показано на рис. 17.2. Следует выбрать: один элемент управления, отсутствие лицензии на использование (в группе Would you like the controls in this project to have a runtime license? нужно выбрать переключатель No runtime license), наличие комментариев в исходных текстах и наличие файлов справки. Установив соответствующие переключатели, щелкните на кнопке Next.

Лицензия на использование

Многие разработчики создают элементы управления как программные продукты, которые они продают. Другие программисты покупают права на использование этих элементов управления в саоих программах. Представим себе, что разработчик Алиса создала просто фантастический элемент управления в виде игральной кости и продала его Бобу, который астроил этот элемент управления в самую лучшую из когда-либо созданных программ для игры в трик-трак. Кэрол приобрела эту игру, и ей чрезвычайно понравился элемент управления в виде игральной кости. Она решает поместить его в разрабатываемую ею игру для детей. Поскольку файл `DIEROLL.OCX` присутствует в дистрибутивном комплекте игры в трик-трак, ничто (кроме этических соображений) не сможет помешать ей сделать это.

Идея проверки наличия лицензии на выполнение очень проста: имеется второй файл (`DIEROLL.LIC`), который, собственно, и содержит лицензию. Без этого файла элемент управления невозможно внедрить в форму или программу, хотя программы, в которые данный элемент управления уже внедрен, будут исправно работать. Алиса предоставила Бобу оба файла, `DIEROLL.OCX` и `DIEROLL.LIC`, но лицензионное соглашение на эту поставку допускает возможность включения в игру в трик-трак только одного файла, `DIEROLL.OCX`. В данном случае Кэрол может сколько угодно восхищаться `DIEROLL.OCX`, и соответствующий элемент управления будет прекрасно работать в игре в трик-трак, но, если она захочет использовать его в собственных разработках, ей придется приобрести соответствующую лицензию у Алисы.

Мастер `AppWizard` предлагает вам организовать подобного рода лицензирование элемента управления в самом начале его создания. Если вы вспомните о лицензировании, когда элемент будет уже почти закончен, придется начать все сначала — создать новый элемент управления с включением опции лицензирования, а потом скопировать в него все ранее разработанные фрагменты программ.

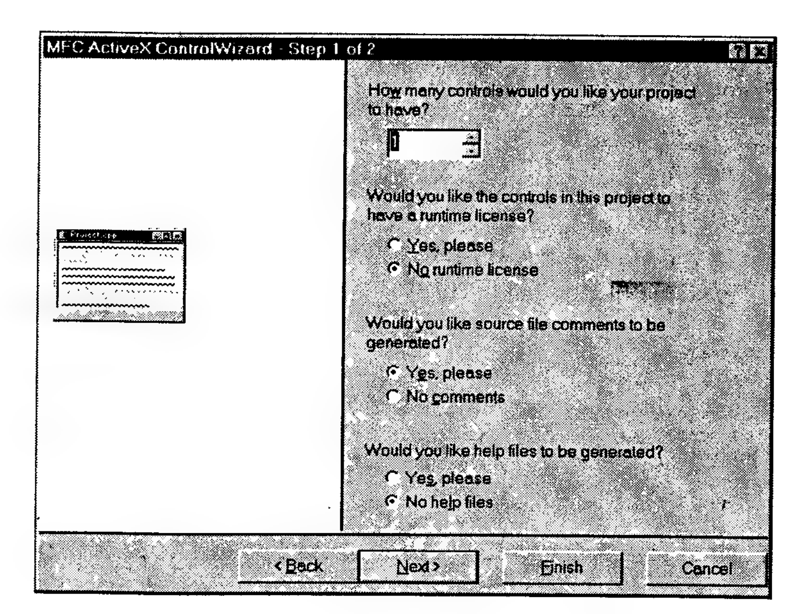


Рис. 17.2. Первый этап настройки мастера `AppWizard` *предназначен для определения основных параметров приложения*

Второй, и последний, этап настройки мастера `AppWizard` предоставляет вам возможность определить функциональные возможности вновь создаваемого элемента управления. Убедитесь, что для создаваемого элемента управления вы установили флажки опций Activate when visible, Available in "Insert Object" dialog и Has an "About" box, как показано на рис. 17.3, а затем щелкните на кнопке Finish. В последнем диалоговом окне настройки `AppWizard` выведет итоговую сводку заданных параметров. Щелкните на кнопке OK, после чего мастер создаст 19 файлов, данные о которых будут помещены в файл проекта, что даст вам возможность работать с нимн, как с единым целым. Все файлы готовы к компиляции, но это лишь пустая оболочка, и теперь ваша задача состоит в том, чтобы ее заполнить.

Текст программ, сгенерированный мастером `AppWizard`

Можно подумать, что девятнадцать файлов — это, пожалуй, уж слишком, но это только на первый взгляд. Созданы всего три класса: `CDierollApp`, `CDierollCtrl` и `CDierollPropPage`. Для этого потребовалось шесть файлов, в число остальных тринадцати входят файл проекта, `make`-файл, файл ресурсов, база данных `ClassWizard` и т.д.

Класс `CDierollApp`

Класс `CDierollApp` — очень маленький класс. Он наследует от класса `COleControlModule` и задает перегрузку его методов `InitInstance()` и `ExitInstance()`, которые в заготовке не выполняют ничего, кроме вызова одноименных методов базового класса. Именно здесь вы найдете `_tlid`, внешний глобальный уникальный идентификатор вашего элемента управления и несколько номеров версий, которые упрощают обновление при распространении новых версий созданного элемента управления. В файле `Dieroll.cpp` значения упомянутых выше идентификаторов определяются в следующих строках.

```
const GUID CDECL BASED_CODE _tlid =
   { 0x914b21a5, 0x7946, 0x11d0, { 0x9b, 0x1, 0, 0x80, 0xc8, 0x1a, 0x39, 0x7c } };
const WORD _wVerMajor = 1;
const WORD _wVerMinor = 0;
```

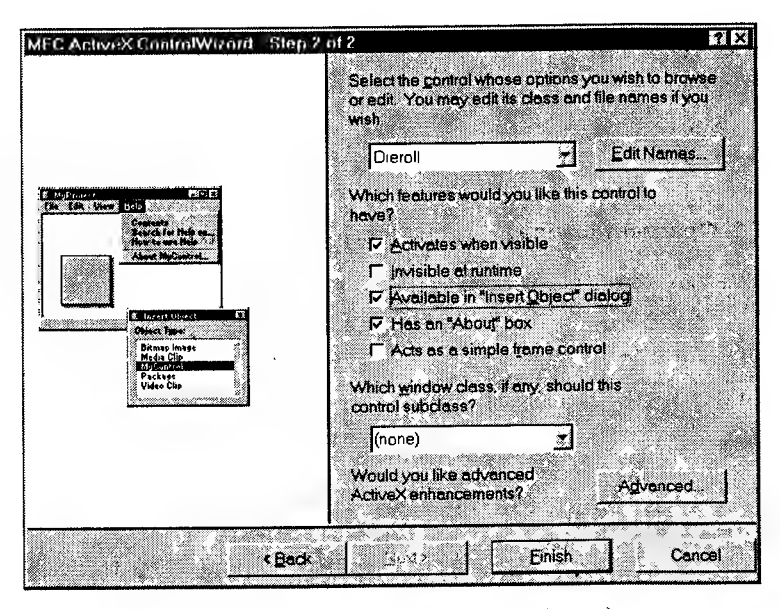


Рис. 17.3. На втором этапе настройки AppWizard *определяется внешний вид и поведение создаваемого элемента управления*

Класс CDierollCtrl

Класс CDierollCtrl наследует от класса COleControl. В нем перегружаются конструктор, деструктор и еще четыре указанные ниже метода.

- OnDraw() (рисует изображение элемента управления)
- DoPropExchange() (осуществляет хранение и инициализацию данных)
- OnResetState() (вызывает повторную инициализацию элемента управления)
- AboutBox() (выводит для элемента управления диалоговое окно About).

Текст этих функций не представляет для нас интереса, чего не скажешь о некоторых добавленных в данный класс картах. Здесь присутствуют пустая карта сообщений, подготовленная для включения в нее новых элементов, и пустая карта диспетчера, подготовленная для внесения в нее свойств и методов, экспортируемых элементом управления.



Назначение карты сообщений разъяснялось в главе 3. Карта диспетчера обсуждалась в главе 16.

Под пустыми картами сообщений и диспетчера появилась новая карта — *карта событий*. Относящиеся к карте событий операторы, которые включены в файл заголовка, представлены в листинге 17.1, а соответствующая ей карта событий в файле реализации — в листинге 17.2.

Листинг 17.1. Фрагмент файла DierollCtl.h — карта событий

```
// Карта событий.
   //{{AFX_EVENT(CDierollCtrl)

      // ВНИМАНИЕ: в данном блоке ClassWizard будет выполнять
      // добавление и удаление функций-членов.

      // НЕ РЕДАКТИРУЙТЕ то, что находится в этом блоке!
   //}}AFX_EVENT
   DECLARE_EVENT_MAP()
```

Листинг 17.2. Фрагмент файла DierollCtl.cpp — карта событий

```
BEGIN_EVENT_MAP(CDierollCtrl, COleControl)
   //{{AFX_EVENT_MAP(CDierollCtrl)

   // ВНИМАНИЕ: в данном блоке ClassWizard будет выполнять
   // добавление и удаление входов карты событий.
   // НЕ РЕДАКТИРУЙТЕ то, что находится в этом блоке!

   //}}AFX_EVENT_MAP
END_EVENT_MAP()
```

Карта событий, как и карта сообщений или карта диспетчера, предназначена для описания связи событий внешней среды с программой. Карта сообщений описывает перехват сообщений, порождаемых действиями пользователя, например выбор команды меню или щелчок на кнопке. Кроме того, она описывает перехват сообщений, посылаемых от одной части приложения к другой. Карта диспетчера управляет запросами на доступ к свойствам или активизацию методов, предоставляемых сервером автоматизации или элементом управления ActiveX. Карта событий управляет извещениями, которыми элемент управления ActiveX обменивается с приложением, его включающим (подробнее речь об этом будет идти ниже в этой же главе).

В файле DierollCtl.cpp имеется еще один фрагмент, на который стоит обратить внимание. Текст его приведен в листинге 17.3.

Листинг 17.3. Фрагмент файла DierollCtl.cpp — вкладки свойств

```
/////////////////////////////////////////////////////////////////////////////
// Property pages
// Вкладки свойств.


// TODO: если необходимо, добавьте сюда новые вкладки свойств.
// Не забывайте увеличивать значение счетчика!

BEGIN_PROPPAGEIDS(CDierollCtrl, 1)
   PROPPAGEID(CDierollPropPage::guid)
END_PROPPAGEIDS(CDierollCtrl)
```

Текст, представленный в листинге 17.3, является частью механизма, который обеспечивает в создаваемом управляющем элементе поддержку страниц свойств — важной части современного интуитивно понятного пользовательского интерфейса.

Класс CDierollPropPages

Класс CDierollPropPages создается ClassWizard для отображения окна свойств. Как и любой иной класс, имеющий в своем составе диалоговое окно, данный класс содержит значительное число компонентов обмена данными. Конструктор инициализирует поля диалогового окна, используя программный код, подготовленный мастером ClassWizard (листинг 17.4).

Листинг 17.4. Файл DierollPpg.cpp — функция CDierollPropPage::CDierollPropPage()

```
CDierollPropPage::CDierollPropPage() :
   COlePropertyPage(IDD, IDS_DIEROLL_PPG_CAPTION)
{
   //{{AFX_DATA_INIT(CDierollPropPage)

   // ВНИМАНИЕ: ClassWizard разместит здесь операторы инициализации переменных.
   // НЕ РЕДАКТИРУЙТЕ то, что находится в этом блоке!
   //}}AFX_DATA_INIT
}
```

Функция DoDataExchange() является посредником при передаче данных между классом CDierollProcPage, который представляет диалоговое окно, содержащее вкладки свойств, и реальными окнами, расположенными на экране пользователя. Текст ее также генерируется мастером ClassWizard (листинг 17.5).

Листинг 17.5. Файл DierollPpg.cpp — функция CDierollPropPage::DoDataExchange()

```
void CDierollPropPage::DoDataExchange(CDataExchange* pDX)
{
   //{{AFX_DATA_MAP(CDierollPropPage)

   // ВНИМАНИЕ: ClassWizard добавит сюда вызовы DDP, DDX и DDV.
   // НЕ РЕДАКТИРУЙТЕ то, что содержится в этом блоке!
   //}}AFX_DATA_MAP
   DDP_PostProcessing(pDX);
}
```

Здесь присутствует, что не удивительно, карта сообщений для класса CDierollPropPage, а также операторы, выполняющие регистрацию (листинг 17.6). К ним система управления ActiveX обращается в тех случаях, когда пользователь выполняет редактирование свойств элемента управления.

Листинг 17.6. Файл DierollPpg.cpp — функция CDierollPropPage::UpdateRegistry()

```
/////////////////////////////////////////////////////////////////////////////
// Initialize class factory and guid
// Инициализация фабрики классов.

IMPLEMENT_OLECREATE_EX(CDierollPropPage, "DIEROLL.DierollPropPage.1",
   0x914b21a8, 0x7946, 0x11d0, 0x9b, 0x1, 0, 0x80, 0xc8, 0x1a, 0x39, 0x7c)

/////////////////////////////////////////////////////////////////////////////
// CDierollPropPage::CDierollPropPageFactory::UpdateRegistry -
// Добавление или удаление записи в системном реестре для CDierollPropPage.

BOOL CDierollPropPage::CDierollPropPageFactory::UpdateRegistry(BOOL bRegister)
```

```
{
   if (bRegister)
      return AfxOleRegisterPropertyPageClass(AfxGetInstanceHandle(),
         m_clsid, IDS_DIEROLL_PPG);
   else
      return AfxOleUnregisterClass(m_clsid, NULL);
}
```

Разработка элемента управления

Как правило, элемент управления имеет внутренние данные (свойства), которые он отображает для пользователя тем или иным способом. Пользователь выполняет некоторые действия, влекущие за собой изменение внутренних данных элемента управления и, возможно, их представления на экране. Некоторые элементы управления представляют пользователю данные, получаемые ими из других источников, таких как базы данных или удаленные файлы. Единственный внутренний параметр, имеющий смысл для элемента управления в виде игральной кости (кроме тех, которые предназначены для определения способа его отображения на экране и будут обсуждаться позднее), — это целое число, принимающее значение от 1 до 6 и определяющее текущее значение, полученное после броска кубика. Позднее наш элемент управления будет отображать свое текущее состояние с помощью узора из точек, как и все игральные кости в реальном мире, но первая версия функции OnDraw() будет просто выводить цифру. Другим упрощением на этапе создания базовой структуры программы будет жесткое задание одной цифры как единственно возможного состояния элемента управления. Текст для имитации случайного броска кости будет добавлен позднее, а пока мы сосредоточимся на обработке действий пользователя.

Отображение текущего значения

Прежде чем состояние будет отображено на экране, элемент управления должен это отображаемое значение иметь. Это означает, что сначала надо включить в элемент управления некоторое свойство, после чего можно будет приступить к созданию подпрограммы его вывода на экран.

Добавление нового свойства

Элемент управления ActiveX может иметь четыре типа свойств.

- Stock (типовые) — это стандартные свойства, предоставляемые любому элементу управления, такие как тип шрифта и цвет. Разработчик должен активизировать типовые свойства элемента управления, но это почти не потребует от него усилий.
- Ambient (свойства окружения) — это свойства среды, в которой "существует" элемент управления, т.е. контейнера, в который элемент управления помещен. Эти свойства не могут быть изменены, но элемент управления может использовать их для настройки собственных свойств. Например, он может устанавливать свой цвет фона в соответствии с цветом фона контейнера.
- Extended (внешние) — это свойства, которые контролируются контейнером. Как правило, к ним относятся размер элемента и его расположение на экране.
- Custom (пользовательские) — это свойства, которые определяются разработчиком элемента управления.

Для хранения текущего состояния создаваемого нами элемента управления добавим к нему с помощью ClassWizard пользовательское свойство Number. С этой целью выполните следующие операции.

1. Выберите команду View⇨ClassWizard, а затем в раскрывшемся окне выберите вкладку Automation.
2. Убедитесь, что в раскрывающемся списке в левой части окна выделен элемент Dieroll (если, конечно, вы не присвоили элементу управления другое имя при настройке мастера AppWizard), а в списке справа выделено имя класса CDierollCtrl.
3. Щелкните на кнопке Add Property, а затем заполните поля раскрывшегося диалогового окна значениями, как показано на рис. 17.4.
4. В поле External Name введите значение Number и обратите внимание на то, какие значения мастер ClassWizard предлагает для полей Variable Name и Notification function.
5. Установите для типа переменной значение short.
6. Для закрытия диалогового окна Add Property щелкните на кнопке OK, а затем еще раз щелкните на кнопке OK для закрытия окна ClassWizard.

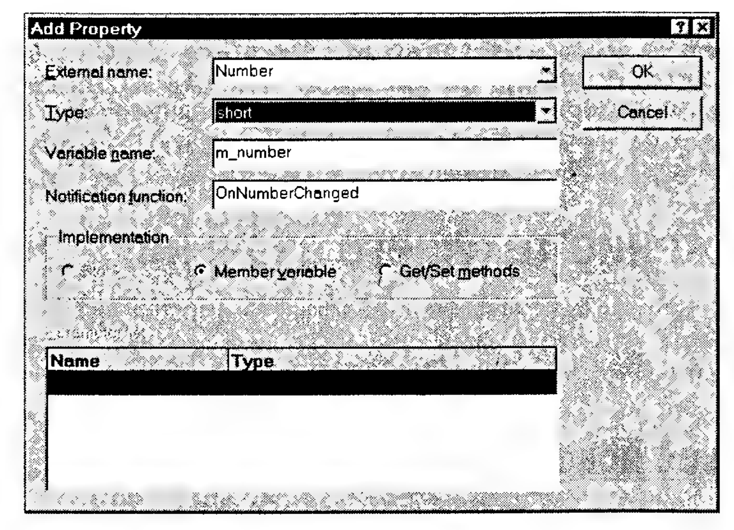


Рис. 17.4. ClassWizard упрощает процедуру добавления пользовательского свойства к создаваемому элементу управления

Прежде чем взяться за подпрограмму, отображающую значение свойства Number, нужно обеспечить установку этого значения. Свойства элементов управления инициализируются в функции DoPropExchange(). Этот метод обеспечивает *живучесть*, т.е. он дает управляющему элементу возможность сохранять свои данные как часть документа приложения-контейнера и восстанавливать их при открытии документа. Если создается новый экземпляр элемента управления, свойства которого не могут быть считаны из файла, то в этом случае им присваиваются значения по умолчанию, обеспечиваемые этим же методом. Для элементов управления не используется метод Serialize().

При генерации приложения мастером AppWizard была создана заготовка метода DoPropExchange() (листинг 17.7).

Листинг 17.7. Файл DierollCtl.cpp — функция CDierollCtrl::DoPropExchange()

```
void CDierollCtrl::DoPropExchange(CPropExchange* pPX)
{
   ExchangeVersion(pPX, MAKELONG(_wVerMinor, _wVerMajor));
   COleControl::DoPropExchange(pPX);


   // TODO: выполните вызов функции PX_ для каждого
   // восстанавливаемого пользовательского свойства.

}
```

Обратите внимание на использование номера версии для контроля того, что файл содержит значения, сохраненные той же версией данного управляющего элемента. Удалите комментарий // ЧТО СДЕЛАТЬ, вставленный мастером AppWizard в качестве напоминания, и поместите на его место следующую строку:

```
PX_Short( pPX, "Number", m_number, (short)3 );
```

Функция PX_Short() является одной из многих предоставленных в ваше распоряжение функций, осуществляющих ввод-вывод данных. Имеется по одной функции на каждый тип свойств, поддерживаемых системой программирования. Функциям передаются следующие аргументы:

- указатель, который был передан функции DoPropExchange();
- внешнее имя свойства, которое вы присвоили ему в диалоговом окне Add Property мастера ClassWizard;
- идентификатор переменной-члена, который вы определили для данного свойства в том же диалоговом окне;
- значение, присваиваемое свойству по умолчанию (для нашего случая мы заменим жестко заданное сейчас значение 3 случайно выбранным значеним несколько позднее).

Существуют следующие функции PX.

```
PX_Blob() (для типа BLOB - binary large object,(двоичный объект))
PX_Bool()
PX_Color() (OLE_COLOR)
PX_Currency()
PX_DATAPATH (CDataPathProperty)
PX_Double()
PX_Float()
PX_Font()
PX_IUnknown() (для типа LPUNKNOWN, указателя интерфейса COM)
PX_Long()
PX_Picture()
PX_Short()
PX_String()
PX_ULong()
PX_UShort()
```

Для одних свойств определить значение по умолчанию очень просто, для других могут потребоваться дополнительные операции. Например, цвет задается с помощью макрокоманды RGB(), которая получает значения для красной, зеленой и синей составляющих в пределах от 0 до 255, а возвращает параметр типа COLORDEF. Допустим, существует некоторое свойство с внешним именем EdgeColor и внутренним именем m_edgecolor, которому по умолчанию необходимо присвоить значение, определяющее серый цвет. Все это можно описать такой строкой:

```
PX_Short( pPX, "EdgeColor", m_edgecolor, RGB(128,128,128) );
```

Элементы управления, имеющие свойства для задания типа шрифта, должны по умолчанию устанавливать шрифт, применяемый контейнером. Для получения имени этого шрифта следует использовать метод AmbientFont() класса COleControl.

Создание подпрограммы вывода изображения на экран

Фрагмент программы, выводящий число на экран, входит в метод OnDraw() класса элемента управления CDierollCtrl. (Элементы управления не имеют классов документов или представлений.) Эта функция автоматически вызывается всякий раз, когда Windows необходимо перерисовать ту часть экрана, в которой располагается данный элемент управления. В листинге 17.8 приведен текст заготовки этой функции, подготовленной мастером AppWizard при создании элемента управления.

Листинг 17.8. Файл DierollCtl.cpp — функция CDierollCtrl::OnDraw()

```
void CDierollCtrl::OnDraw(CDC* pdc, const CRect& rcBounds,
    const CRect& rcInvalid)
{

    // TODO: замените следующие ниже строки собственным
    // текстом, который программирует вывод изображения на экран.
    pdc->FillRect(rcBounds,
    CBrush::FromHandle((HBRUSH)GetStockObject(WHITE_BRUSH)));
    pdc->Ellipse(rcBounds);
}
```

Как указывалось в главе 5, управляющая программа передает функции OnDraw() контекст устройства, в котором строится изображение, объект класса CRect, описывающий участок, отведенный данному элементу управления, и еще один объект CRect, описывающий участок, который следует перерисовать. В тексте, приведенном в листинге 17.8, выполняется вывод белого прямоугольника в пределах rcBounds, а затем внутри этого прямоугольника рисуется эллипс, выполненный цветом, принятым по умолчанию для объектов переднего плана. Вывод на экран белого прямоугольника можно сохранить, но вместо построения в нем эллипса следует вывести изображение символа, соответствующего значению свойства Number. Для этого необходимо заменить последний оператор в заготовке функции OnDraw() на следующие:

```
CString val; // Символ, представляющий значение переменной типа short.
val.Format("%i",m_number);
pdc->ExtTextOut( 0, 0, ETO_OPAQUE, rcBounds, val, NULL );
```

В этом фрагменте программы значение переменной m_number, имеющей тип short (связанной нами в диалоговом окне Add Property со свойством Number), преобразуется с помощью новой функции CString::Format() в значение переменной val, имеющей тип CString. (Это иллюстрирует, как, используя новейшие методы программировании на C++, избавиться от необходимости обращения к функции sprintf().) Функция ExtTextOut() выводит на экран в прямоугольнике rcBounds текст — символ, содержащийся в val. Выводимое число всегда будет равно 3, поскольку именно так на данный момент определено значение свойства Number.

Если вы хотите удостовериться в том, как просто создать элемент управления ActiveX, который уже что-то способен выполнять, можете прямо сейчас оттранслировать вновь созданный элемент управления, а затем провести тестирование его работы. В отличие от других типов приложений ActiveX, элементы управления не могут быть запущены на выполнение с целью их регистрации как самостоятельные задания. Оттранслируйте проект и устраните

возможные мелкие ошибки, сделанные при вводе текста програм. Для организации окна контейнера тестирования элементов управления, показанного на рис. 17.5, выберите команду Tools⇨ActiveX Control Test Container.

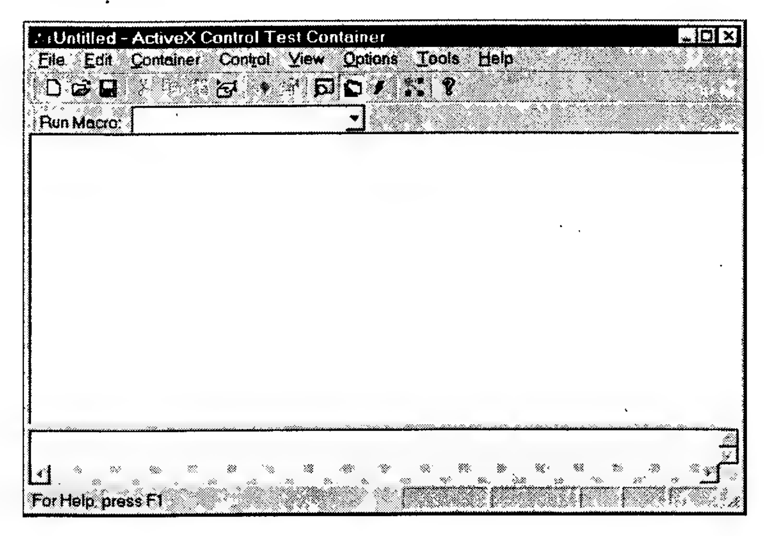


Рис. 17.5. Контейнер тестирования элементов управления ActiveX является идеальным инструментом для тестирования разрабатываемых элементов управления

На заметку

Если в окне Visual Studio в меню Tools отсутствует команда ActiveX Control Test Container, ее можно поместить в указанное меню, выполнив следующие операции.

1. Выберите команду Tools⇨Customize.
2. Щелкните на корешке вкладки Tools.
3. Просмотрите список инструментов и убедитесь, что ActiveX Control Test Container в нем не присутствует.
4. Перейдите в конец списка и сделайте двойной щелчок мышью на пустой строке.
5. Введите в пустое поле значение `ActiveX Control Test Container` и нажмите клавишу <Enter>.
6. Щелкните на кнопке "..." справа от окна Command и найдите папку `BIN`, расположенную в папке `Visual Studio` на имеющемся у вас компакт-диске или на том жестком диске, на котором инсталлирован Visual C++. Выделите программу `tstcon32.exe` и щелкните на кнопке OK для завершения поиска. В большинстве случаев путь к этой программе будет следующим: `C:\Program Files\Microsoft Visual Studio\Common\Tools\ TSTCON32.EXE` (хотя в вашей системе он может оказаться и другим).
7. Щелкните на стрелке вправо рядом с окном Initial Directory и выберите в раскрывшемся списке значение `Target Directory`.
8. Убедитесь, что все три флажка опций в нижней части окна папки сброшены.
9. Щелкните на кнопке Close.

После того как вы один раз установите тестовый контейнер, проводить повторные инсталляции больше не потребуется. Поместив тестовый контейнер в среду `Visual Studio` так, как рекомендовано выше, вы максимально упростите загрузку и тестирование создаваемого приложения с помощью этого инструмента.

В окне тестового контейнера выберите команду Edit⇨Insert New Control, а затем в раскрывшемся списке выберите значение Dieroll Control. На рис. 17.6 созданный нами элемент управления выглядит, как белый прямоугольник, в котором выведена маленькая цифра 3. Имеется возможность перетаскивать наш элемент управления внутри контейнера и изменять его размеры, но цифра 3 в верхнем левом углу будет оставаться неизменной. Далее мы доработаем программу так, чтобы выводимое число изменялось, если пользователь сделал на элементе управления щелчок мышью.

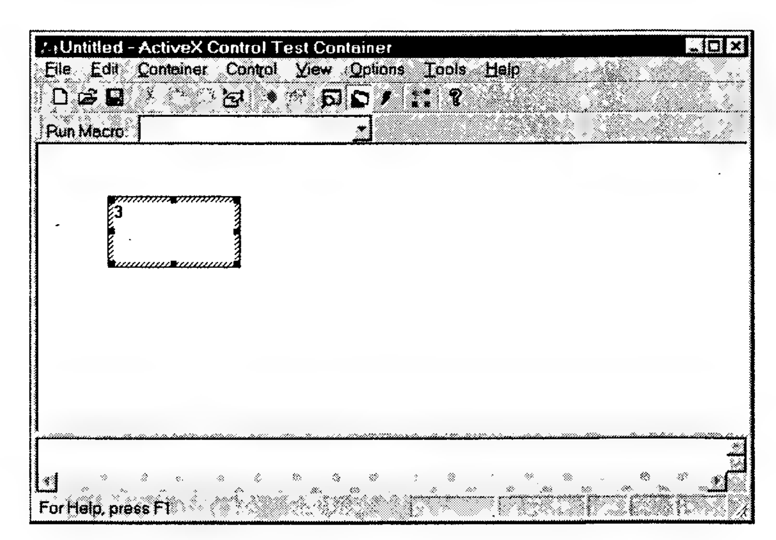


Рис. 17.6. Добавив одно свойство и изменив две функции, мы превратили пустую оболочку в элемент управления, отображающий цифру 3

Имитация броска кости в ответ на щелчок мышью

Когда пользователь щелкает на элементе управления мышью, элемент должен выполнить две операции — проинформировать контейнер о том, что пользователь сделал щелчок мышью, и имитировать бросок кости с отображением вновь полученного значения.

Уведомление контейнера

Решая эту задачу, мы впервые встречаемся с использованием *событий* для уведомления контейнера. События — это то, что элементы управления используют для уведомления контейнера о действиях, выполненных пользователем. Точно так же, как существуют типовые свойства, существуют и типовые события. Мастер AppWizard автоматически подготавливает подпрограммы обработки следующих событий:

- Click (щелчок) используется для извещения контейнера о том, что пользователь сделал щелчок мышью;
- DblClick (двойной щелчок) используется для извещения контейнера о том, что пользователь сделал двойной щелчок мышью;

- Error (ошибка) используется для извещения контейнера о том, что произошла ошибка, которая не может быть обработана с помощью других событий;
- KeyDown (нажатие клавиши) используется для извещения контейнера о том, что нажата клавиша клавиатуры;
- KeyPress (ввод клавиши) используется для извещения контейнера о том, что имело место нажатие и отпускание клавиши клавиатуры;
- KeyUp (отпускание клавиши) используется для извещения контейнера о том, что клавиша клавиатуры отпущена;
- MouseDown (нажатие кнопки мыши) используется для извещения контейнера о том, что нажата кнопка мыши;
- MouseMove (перемещение мыши) используется для извещения контейнера о том, что указатель мыши перемещается над элементом управления;
- MouseUp (отпускание кнопки мыши) используется для извещения контейнера о том, что кнопка мыши отпущена.

Самый лучший способ известить контейнер о том, что пользователь сделал щелчок на элементе управления, — это послать ему типовое событие Click. Прежде всего с помощью ClassWizard необходимо добавить это событие в элемент управления. Выполните следующие операции.

1. Раскройте на экране окно ClassWizard, выбрав команду View⇨ClassWizard, а затем щелкните на корешке вкладки ActiveX Elements. Убедитесь, что выделен класс CDierollCtrl.
2. Щелкните на кнопке Add Event и заполните поля в диалоговом окне Add Event так, как показано на рис. 17.7.
3. Нам требуется добавить событие с внешним именем Click. Выберите его в раскрывающемся списке и обратите внимание на то, что поле внутреннего имени заполнено значением FireClick.
4. Для добавления события щелкните на кнопке OK — и работа будет выполнена.

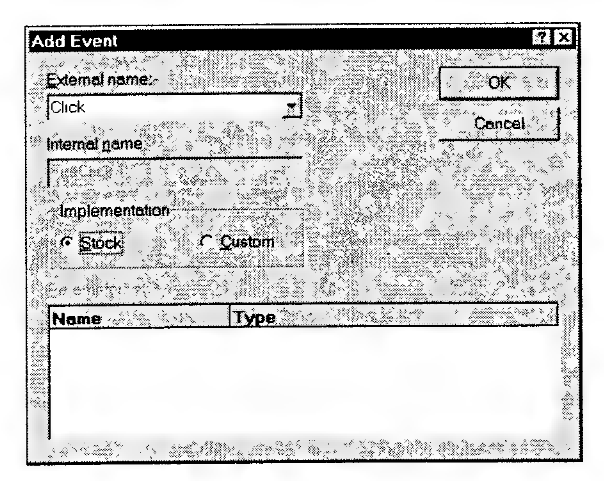
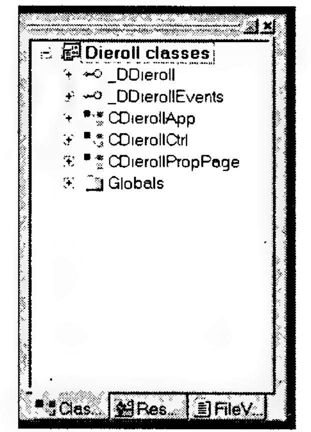


Рис. 17.7. Мастер ClassWizard поможет вам добавить событие в элемент управления



Вероятно, вы обратили внимание на то, что на вкладке ClassView появились две новые пиктограммы, напоминающие ручки регулировки. Щелкните на знаке "+" перед элементом DDierollEvents, и вы сможете увидеть иа схеме событие Click, показанное в качестве события данного приложения (рис. 17.8).

Теперь, если пользователь сделает щелчок на управляющем элементе, класс контейнера будет об этом уведомлен. Так что если вы, например, разрабатываете игру в трик-трак, то контейнер сможет отреагировать на щелчок, используя новое значение на игральной кости для вычисления возможиых ходов или выполнения других, специфических для этой игры заданий.

Рис. 17.8. События отображаются на вкладке ClassView так же, как и классы

Второй аспект процесса обработки щелчка пользователя на управляющем элементе предусматривает выполнение действий, имитирующих бросок игральной кости и вывод на экран нового значения. Не стоит удивляться тому, что ClassWizard предоставит нам свою помощь и при решении этой задачи. Когда пользователь выполняет щелчок на управляющем элементе, это событие перехватывается элементом в карте сообщений, как и в случае обычного приложения. В среде разработки окно ClassWizard должно быть все еще открыто. Если это не так, откройте его и выполните следующие операции.

1. На этот раз выберите вкладку Message Maps и убедитесь, что в окне списка Class Name выделено имя класса данного элемента управления, CDierollCtrl.

2. В списке Messages найдите сообщение WM_LBUTTONDOUN, которое генерируется системой Windows в тех случаях, когда на управляющем элементе выполняется щелчок левой кнопкой мыши.

3. Для добавления функции, которая будет автоматически вызываться при появлении этого сообщения (т.е. после щелчка пользователя на элементе управления), щелкните на кнопке Add Function. Эта функция обязательно должна иметь имя OnLButtonDown(), поэтому ClassWizard не будет выводить диалоговое окно с предложением подтвердить принятие предложенного им имени.

4. Мастер ClassWizard создаст заготовку функции OnLButtonDown(). Щелчок на кнопке Edit Code закроет окно ClassWizard, и вы сможете отредактировать текст сгенерированной функции OnLButtonDown(). Вот как она выглядит сразу после создания.

   ```
   void CDierollCtrl::OnLButtonDown(UINT nFlags, CPoint point)
   {
       // TODO: поместите сюда текст для обработки
       // сообщения и/или вызов функции по умолчанию.

       COleControl::OnLButtonDown(nFlags, point);
   }
   ```

5. Замените комментарий // TODO вызовом новой функции Roll(), которую мы напишем в следующем разделе. Эта функция будет возвращать случайное целое значение от 1 до 6:

   ```
   m_number = Roll();
   ```

6. Потребовать обязательную перерисовку элемента управления можно, добавив в функцию следующую строку:

   ```
   InvalidateControl();
   ```

7. В конце функции сохраните вызов метода базового класса COleControl::OnLButtonDown(), который будет выполнять все остальное, что необходимо при обработке щелчка мышью.

Имитация броска игральной кости

Для того что добавить метод Roll() в класс CDierollCtrl, сделайте щелчок правой кнопкой мыши на имени класса CDierollCtrl на панели ClassView, а затем выберите команду Add Member Function в раскрывшемся контекстном меню. Как показано на рис. 17.9, функция Roll() должна быть открытой, не иметь аргументов и возвращать значение типа short.

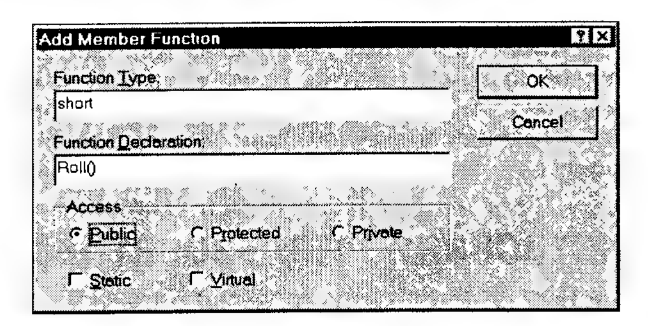


Рис. 17.9. Для ускорения решения рутинных задач используйте диалоговое окно Add Member Function

Что должна выполнять функция Roll()? Она должна вычислять случайное целое число, лежащее в диапазоне от 1 до 6. В языке C++ имеется стандартная функция rand(), возвращающая случайное целое число в диапазоне от 0 до RAND_MAX. Деление возвращенного ею значения на RAND_MAX+1 даст в результате положительное число, которое всегда будет меньше 1. Если умножить это частное на 6, мы получим положительное число, которое всегда будет меньше 6. Другими словами, целая часть этого числа будет в диапазоне от 0 до 5. Добавление к целой части единицы даст нам требуемый результат: целое число в диапазоне от 1 до 6. Соответствующий текст программы представлен в листинге 17.9.

Листинг 17.9. Файл DierollCtl.cpp — функция CDierollCtrl::Roll()

```
short CDierollCtrl::Roll(void)
{
   double number = rand();
   number /= RAND_MAX + 1;
   number *= 6;
   return (short)number + 1;
}
```

На заметку

Если значение RAND_MAX+1 не будет кратно 6, данная функция будет возвращать малые значения несколько чаще, чем большие. Стандартное значение RAND_MAX — 32 767. Это означает, что каждый из результатов 1 и 2 будет в среднем возвращен 5 462 раза из 32 767 вызовов функции, в то время как значения от 3 до 6 будут возвращены по 5 461 разу. Проигнорируем эту неточность.

В некоторых подпрограммах имитации броска игральной кости вместо данного алгоритма используется функция взятия модуля. Однако такой вариант имеет куда худшие показатели. Меньшие значения будут появляться заметно реже остальных. Используемый нами алгоритм дает распределение случайных чисел, более близкое к равномерному.

Генератор случайных чисел должен быть проинициализирован до того, как он будет использован впервые. Традиционно в качестве исходного значения для генератора используется текущее время. В функцию DoPropExchange() до вызова функции PX_Short() добавьте следующий оператор:

```
srand( (unsigned)time( NULL ) );
```

Вместо жестко закодированного исходного значения, равного 3, обратимся к функции Roll(), которая вернет нам случайное число. Измените вызов функции PX_Short() так, как показано ниже:

```
PX_Short( pPX, "Number", m_number, Roll());
```

Оттранслируйте приложение и проверьте работу созданного элемента управления в контейнере для тестирования. При выполнении на элементе управления щелчков мышью отображаемая цифра меняется после каждого щелчка. Поэкспериментируйте с только что созданным приложением. Попадаются ли значения, меньшие единицы или большие шести? Отмечены ли вами какие-либо странности в поведении элемента управления?

Совершенствование пользовательского интерфейса

Теперь, когда основные функциональные возможности элемента управления нами уже обеспечены, пришло время сделать его внешний вид более привлекательным. Назначим нашему элементу управления некоторую пиктограмму, а значение будем выводить не одной цифрой, а соответствующим количеством точек.

Растровая пиктограмма

Поскольку у некоторых из вас после окончания разработки этого элемента управления может возникнуть желание включить его в Control Palette Visual Basic или Visual C++, необходимо создать пиктограмму, которая и будет его там представлять. Собственно говоря, мастер AppWizard уже назначил элементу управления некую пиктограмму, но она представляет собой всего лишь логотип MFC и не отражает особенностей нашего приложения. Более подходящую пиктограмму можно создать с помощью Visual Studio. В окне Project Workspace щелкните на корешке вкладки ResourceView, а затем щелкните на знаке "+" перед элементом Bitmap, после чего сделайте двойной щелчок на элементе IDB_DIEROLL. Теперь вам предоставлена возможность редактировать пиктограмму, определяя пиксели один за другим. На рис. 17.10 показан образец подходящей пиктограммы. Начиная с этого момента, всякий раз, когда вы будете загружать элемент управления Dieroll в тестовый контейнер, пиктограмма этого приложения будет помещаться на панель инструментов контейнера.

Отображение точек

Следующим шагом на пути создания элемента управления, имитирующего игральную кость, будет придание этому элементу соответствующего внешнего вида. Создание прекрасного трехмерного изображения, предусматривающего отображение, кроме основной грани, еще и частей других граней, выходит далеко за рамки иллюстративного примера данной главы. Но построение изображения из соответствующего количества точек будет вполне уместно.

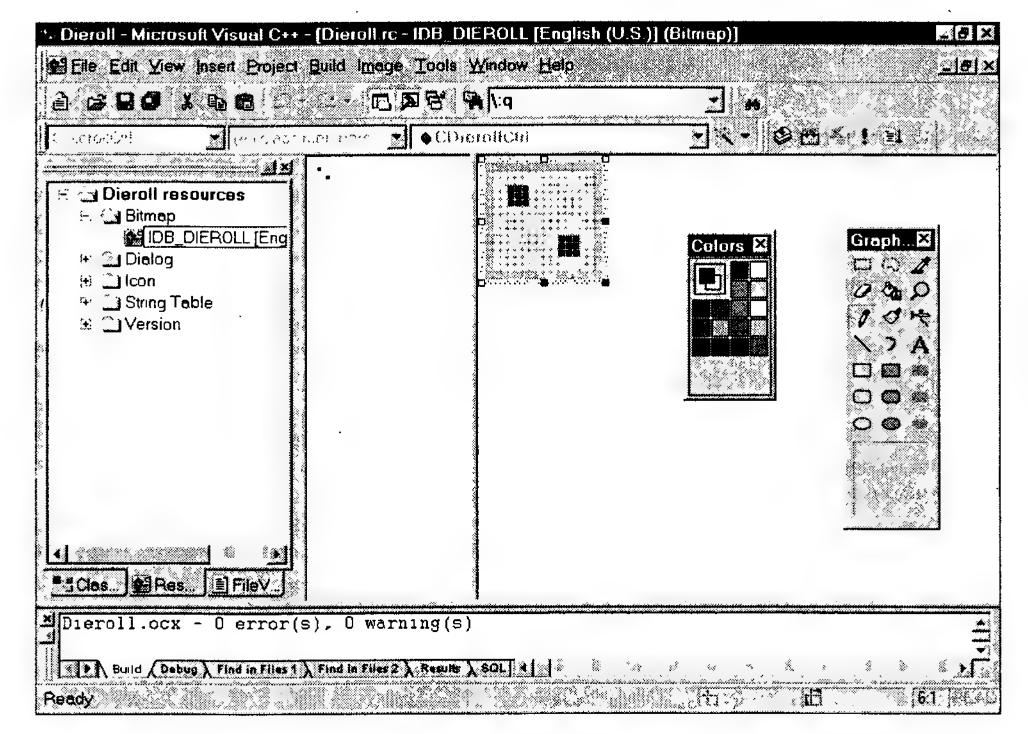


Рис. 17.10. Инструмент ResourceView в Visual C++ позволяет создать собственную пиктограмму, которая будет впоследствии помещена в Control Palette Visual Basic

Первым шагом будет включение в функцию OnDraw() оператора switch. Превратите в комментарий три строки, прежде формировавшие изображение, а затем добавьте оператор switch. После этого функция OnDraw() должна приобрести вид, представленный в листинге 17.10.

Листинг 17.10. Файл DierollCtl.cpp — функция CDierollCtrl::OnDraw()

```
void CDierollCtrl::OnDraw(
   CDC* pdc, const CRect& rcBounds, const CRect& rcInvalid)
{
   pdc->FillRect(rcBounds,
      CBrush::FromHandle((HBRUSH)GetStockObject(WHITE_BRUSH)));
// CString val; //Символьное представление значения типа short.
// val.Format("%i",m_number);
// pdc->ExtTextOut( 0, 0, ETO_OPAQUE, rcBounds, val, NULL );

   switch(m_number)
   {
   case 1:
      break;
   case 2:
      break;
   case 3:
      break;
   case 4:
      break;
   case 5:
      break;
   case 6:
```

```
            break;
        }
    }
```

Теперь все, что осталось сделать, — это добавить текст программ, который в случае оператора case 1: будет рисовать одну точку, в случае оператора case 2: — две точки и т.д. Если у вас под рукой есть настоящая игральная кость, присмотритесь к ней повнимательней. Ширина каждой из точек составляет приблизительно одну четверть ширины всей грани. Точки, расположенные вдоль края, находятся от него на расстоянии примерно в одну шестнадцатую ширины грани. Все значения, кроме 6, можно вписать в шаблон, созданный для значения 5, например единственная точка для значения 1 располагается так же, как и центральная точка в значении 5.

Второй параметр функции OnDraw(), rcBounds, является ссылкой на экземпляр класса CRect, который описывает прямоугольник, занимаемый изображением элемента управления. Он содержит переменные-члены и функции, которые возвращают координаты верхнего левого угла, ширину и высоту изображения элемента управления. Текст функции, который был создан мастером AppWizard по умолчанию для отображения в прямоугольнике эллипса, вызывал функцию CDC::Ellipse(). В нашем варианте функции OnDraw() также будет выполняться вызов функции Ellipse(), причем ей будут передаваться координаты маленького прямоугольника внутри большого прямоугольника всего элемента управления. Текст нашей подпрограммы будет более читабелен, а выполняться она будет чуточку быстрее, если мы будем работать с единицами, составляющими одну шестнадцатую от общей ширины или высоты элемента управления. Тогда размер каждой из точек будет равен четырем единицам по высоте и ширине. Вставьте следующие строки перед оператором switch:

```
int Xunit = rcBounds.Width()/16;
int Yunit = rcBounds.Height()/16;

int Top = rcBounds.top;
int Left = rcBounds.left;
```

Прежде чем нарисовать окружность с помощью вызова функции Ellipse(), необходимо выбрать инструмент, которым будет выполняться рисование. Поскольку наши окружности должны быть закрашенными, их необходимо рисовать кистью. С помощью приведенных ниже операторов создается кисть и контексту устройства pdc передается указание использовать ее, причем указатель на старую кисть сохраняется, так что его впоследствии можно будет восстановить:

```
CBrush Black;
Black.CreateSolidBrush(RGB(0x00,0x00,0x00)); //Сплошная черная кисть.
CBrush* savebrush = pdc->SelectObject(&Black);
```

После оператора switch для восстановления старой кисти следует поместить такой оператор:

```
pdc->SelectObject(savebrush);
```

Теперь все готово к тому, чтобы добавить операторы в каждый из блоков case для рисования необходимого числа точек. Например, значения 2, 3, 4, 5 и 6 требуют наличия точки в верхнем левом углу. Эта точка должна будет располагаться в прямоугольнике, который начинается на единицу дискретности вправо и вниз от левого верхнего угла изображения элемента управления и простирается на пять единиц вправо и на пять единиц вниз. Вызов функции Ellipse() для рисования этой точки будет иметь следующий вид:

```
pdc->Ellipse(Left+Xunit, Top+Yunit,
        Left+5*Xunit, Top + 5*Yunit);
```

Координаты всех остальных точек определяются сходным образом. Окончательный вид оператора switch показан в листинге 17.11.

Листинг 17.11. Файл `DierollCtl.cpp` **— фрагмент функции** `CDierollCtrl::OnDraw()`

```
switch(m_number)
   {
   case 1:
      pdc->Ellipse(Left+6*Xunit, Top+6*Yunit,
            Left+10*Xunit, Top + 10*Yunit); //Центральная.
      break;
   case 2:
      pdc->Ellipse(Left+Xunit, Top+Yunit,
            Left+5*Xunit, Top + 5*Yunit); //Верхняя левая.
      pdc->Ellipse(Left+11*Xunit, Top+11*Yunit,
            Left+15*Xunit, Top + 15*Yunit); //Нижняя правая.
      break;
   case 3:
      pdc->Ellipse(Left+Xunit, Top+Yunit,
            Left+5*Xunit, Top + 5*Yunit); //Верхняя левая.
      pdc->Ellipse(Left+6*Xunit, Top+6*Yunit,
            Left+10*Xunit, Top + 10*Yunit); //Центральная.
      pdc->Ellipse(Left+11*Xunit, Top+11*Yunit,
            Left+15*Xunit, Top + 15*Yunit); //Нижняя правая.
      break;
   case 4:
      pdc->Ellipse(Left+Xunit, Top+Yunit,
            Left+5*Xunit, Top + 5*Yunit); //Верхняя левая.
      pdc->Ellipse(Left+11*Xunit, Top+Yunit,
            Left+15*Xunit, Top + 5*Yunit); //Верхняя правая.
      pdc->Ellipse(Left+Xunit, Top+11*Yunit,
            Left+5*Xunit, Top + 15*Yunit); //Нижняя левая.
      pdc->Ellipse(Left+11*Xunit, Top+11*Yunit,
            Left+15*Xunit, Top + 15*Yunit); //Нижняя правая.
      break;
   case 5:
      pdc->Ellipse(Left+Xunit, Top+Yunit,
            Left+5*Xunit, Top + 5*Yunit); //Верхняя левая.
      pdc->Ellipse(Left+11*Xunit, Top+Yunit,
            Left+15*Xunit, Top + 5*Yunit); //Верхняя правая.
      pdc->Ellipse(Left+6*Xunit, Top+6*Yunit,
            Left+10*Xunit, Top + 10*Yunit); //Центральная.
      pdc->Ellipse(Left+Xunit, Top+11*Yunit,
            Left+5*Xunit, Top + 15*Yunit); //Нижняя левая.
      pdc->Ellipse(Left+11*Xunit, Top+11*Yunit,
            Left+15*Xunit, Top + 15*Yunit); //Нижняя правая.
      break;
   case 6:
      pdc->Ellipse(Left+Xunit, Top+Yunit,
            Left+5*Xunit, Top + 5*Yunit); //Верхняя левая.
      pdc->Ellipse(Left+11*Xunit, Top+Yunit,
            Left+15*Xunit, Top + 5*Yunit); //Верхняя правая.
      pdc->Ellipse(Left+Xunit, Top+6*Yunit,
            Left+5*Xunit, Top + 10*Yunit); //Средняя левая.
      pdc->Ellipse(Left+11*Xunit, Top+6*Yunit,
            Left+15*Xunit, Top + 10*Yunit); //Средняя правая.
      pdc->Ellipse(Left+Xunit, Top+11*Yunit,
            Left+5*Xunit, Top + 15*Yunit); //Нижняя левая.
      pdc->Ellipse(Left+11*Xunit, Top+11*Yunit,
            Left+15*Xunit, Top + 15*Yunit); //Нижняя правая.
      break;
   }
```

Еще раз оттранслируйте наше приложение и проверьте его работу в контейнере для тестирования. Вашему взору должно предстать изображение элемента управления, похожее на то, которое показано на рис. 17.11 и которое на самом деле выглядит, как игральная кость!

Если вы обладаете острым зрением или если размеры получившейся игральной кости невелики, вы сможете заметить, что узор из точек располагается несколько не по центру. Это происходит потому, что высота и ширина элемента управления имеют размеры, не кратные 16. Например, если функция `Width()` возвращает значение 31, то значение `Xunit` будет равно 1. И все точки будут расположены между позициями 0 и 16, оставляя справа пустую полосу. К счастью, ширина элемента управления обычно намного больше 31 пикселя, и поэтому ассимметрия не столь заметна.

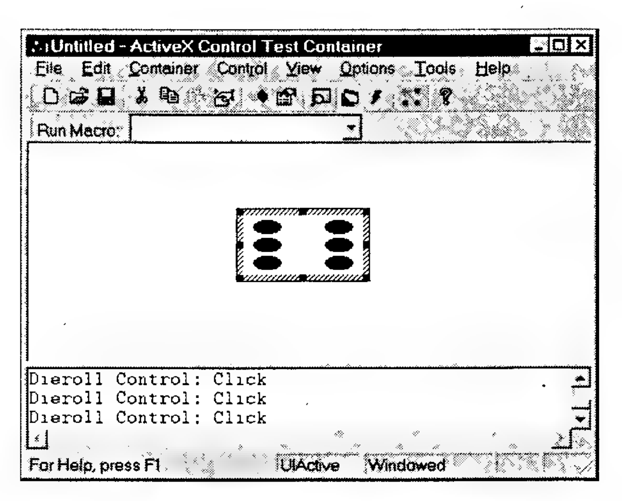


Рис. 17.11. Созданный нами элемент управления теперь выглядит, как игральная кость

Для исправления этой ошибки выполним центрирование точек в изображении элемента управления. Найдите строки программы, в которых вычисляются значения `Xunit` и `Yunit`, а затем добавьте к ним новые строки, показанные в листинге 17.12.

Листинг 17.12. Файл `DierollCtl.cpp` — корректировка значений `Xunit` и `Yunit`

```
//Точки имеют 4 единицы в высоту и ширину, располагаясь на одну единицу от края.
int Xunit = rcBounds.Width()/16;
int Yunit = rcBounds.Height()/16;
int Xleft = rcBounds.Width()%16;
int Yleft = rcBounds.Height()%16;

// Выравнивание верхнего левого угла за счет остатков от деления.
int Top = rcBounds.top + Yleft/2;
int Left = rcBounds.left + Xleft/2;
```

Переменные `Xleft` и `Yleft` содержат “избыток” пикселей в направлениях `X` и `Y`. Увеличивая значения `Top` и `Left` на половинное значение избытка пикселей, мы выполняем центрирование всего изображения, не внося изменений в остальную часть текста программы.

Окна свойств

Элементы управления ActiveX могут иметь окна свойств, которые предоставляют пользователю возможность устанавливать свойства элементов управления, не внося изменений в контейнерное приложение. (Окна и вкладки свойств обсуждались нами в главе 12.) Окна свойств определяются, как диалоговые окна, причем имеются заранее написанные вкладки для задания шрифта, определения цветов и других общих свойств. Для создаваемого нами элемента управления имеет смысл определить вкладки для следующих свойств.

- Флажок для индикации способа отображения значения: в виде цифры или в виде точечного узора
- Цвет объектов переднего плана
- Цвет фона

На заметку

Очень легко ошибиться в том, что, собственно, представляет собой одна страница свойств: является ли она одной из вкладок диалогового окна или это весь набор вкладок диалогового окна? К каждой из вкладок окна свойств применим термин *страница* (page) свойств, а ко всей совокупности вкладок окна свойств применим термин *таблица* (sheet) свойств. Каждая из вкладок определяется в программе как диалоговое окно, а затем с помощью ClassWizard осуществляется связывание значений в этом диалоговом окне с переменными-членами.

Цифры или точки

Позволить пользователю выбрать способ отображения текущего значения (цифрой или точками) — это относительно просто. Достаточно добавить в приложение свойство, которое определяло бы текущий выбор, а затем проанализировать значение этого свойства в функции OnDraw(). Устанавливать значение этого свойства пользователь мог бы на соответствующей вкладке свойств.

Прежде всего с помощью ClassWizard добавим новое свойство. Вот как это следует сделать: раскройте окно ClassWizard и перейдите на вкладку Automation. Убедитесь, что в списке выбран класс CDierollCtrl, а затем щелкните на кнопке Add Property. В диалоговом окне Add Property укажите внешнее имя Dots и внутреннее имя m_dots. Выберите тип представления BOOL, поскольку для свойства Dots достаточно иметь два значения: либо TRUE, либо FALSE. Определите это новое свойство как переменную-член (с прямым доступом) данного класса. Завершите работу в окне Add Property, щелчкнув на кнопке OK, а затем закройте окно ClassWizard, еще раз щелкнув на кнопке OK. Указанная переменная-член добавлена к описанию класса, выполнено обновление карты диспетчера и создана заготовка функции уведомления OnDotsChange().

Для инициализации свойства Dots и сохранения его значения в документе добавьте в функцию DoPropExchange(), сразу после вызова функции PX_Short(), следующую строку:

```
PX_Bool( pPX, "Dots", m_dots, TRUE);
```

Инициализация свойства Dots значением TRUE обеспечивает по умолчанию отображение значения элемента управления точечным узором.

В функции OnDraw() удалите символы комментария из строк, обеспечивающих вывод значения цифрой. Поместите эти строки в оператор if так, чтобы цифра отображалась в тех случаях, когда переменная m_dots имеет значение FALSE, а точечный узор отображался в тех случаях, когда значение этой переменной равно TRUE. В результате текст функции должен выглядеть так, как представлено в листинге 17.13.

Листинг 17.13. Файл DierollCtl.cpp — функция CDierollCtrl::OnDraw()

```
void CDierollCtrl::OnDraw(
      CDC* pdc, const CRect& rcBounds, const CRect& rcInvalid)
{
   pdc->FillRect(rcBounds,
      CBrush::FromHandle((HBRUSH)GetStockObject(WHITE_BRUSH)));

   if (!m_dots)
   {
      CString val; //Символьное представление значения типа short.
      val.Format("%i",m_number);
      pdc->ExtTextOut( 0, 0, ETO_OPAQUE, rcBounds, val, NULL );
   }
   else
   {
      //Точки имеют ширину и высоту в 4 единицы и отстоят на 1 единицу от края.
      int Xunit = rcBounds.Width()/16;
      int Yunit = rcBounds.Height()/16;
      int Xleft = rcBounds.Width()%16;
      int Yleft = rcBounds.Height()%16;

      // Выравнивание верхнего левого угла с учетом избытка длины.
      int Top = rcBounds.top + Yleft/2;
      int Left = rcBounds.left + Xleft/2;

      CBrush Black;
      Black.CreateSolidBrush(RGB(0x00,0x00,0x00)); //Сплошная черная кисть.

      CBrush* savebrush = pdc->SelectObject(&Black);

      switch(m_number)
      {
      case 1:
            ...
      }
      pdc->SelectObject(savebrush);
   }
}
```

Чтобы обеспечить пользователю возможность устанавливать значение свойства Dots, необходимо создать соответствующую вкладку свойств. Выполните с этой целью следующие операции.

1. В окне Project Workspace выберите вкладку ResourceView, а затем щелкните на знаке "+", стоящем перед элементом Dialog.
2. Для нашего элемента управления определены два диалоговых окна: первое из них — окно About, а второе — окно свойств. Сделайте двойной щелчок на элементе IDD_PROPPAGE. Мастер AppWizard создаст заготовку окна свойств с единственным элементом управления — статическим текстом TODO: Place controls to manipulate properties of Dieroll Control on this dialog (ВЫПОЛНИТЬ: вставьте в это диалоговое окно элементы управления для настройки свойств Dieroll).
3. Удалите статический текст с напоминанием, для чего выделите его и нажмите клавишу <Del>.
4. Перетащите с панели инструментов Control Palette в создаваемое диалоговое окно пиктограмму флажка опции. Выберите команду View⇨Properties и зафиксируйте диалоговое окно Properties на отведенном ему месте.
5. В поле Caption введите значение Display Dot Pattern (Отображать точечный узор) и укажите в качестве идентификатора ресурса значение IDC_DOTS, как показано на рис. 17.12.

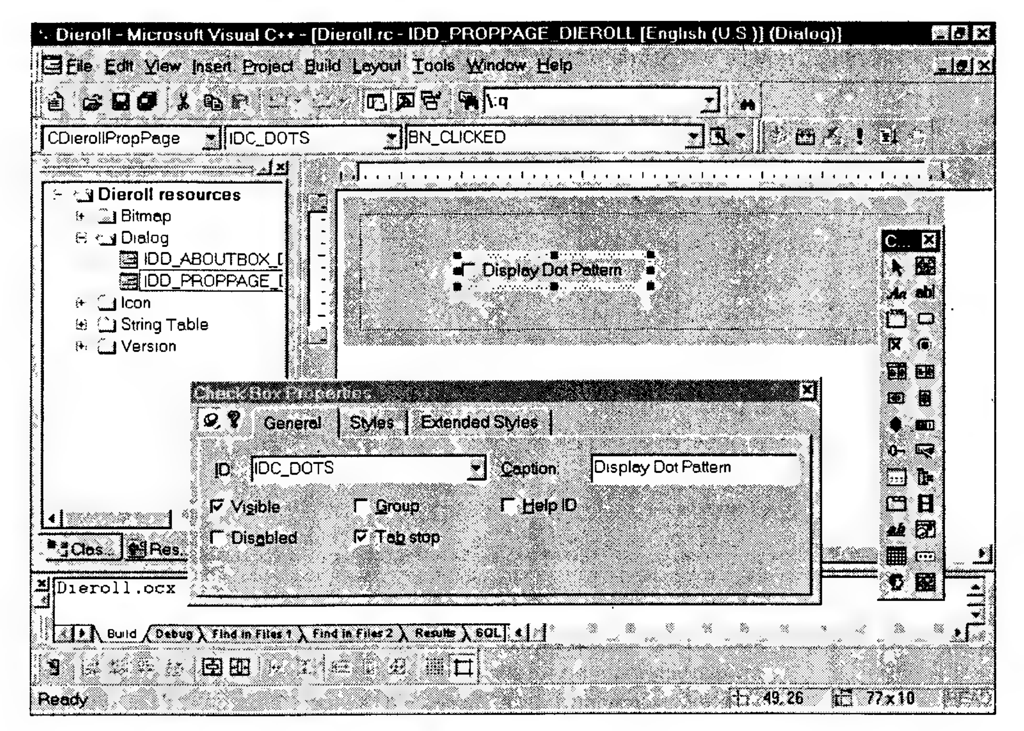


Рис. 17.12. Вкладка свойств для элемента управления ActiveX создается так же, как и любое другое диалоговое окно

Когда пользователь выведет на экран эту вкладку свойств и сделает щелчок для установки или сброса флажка опции, это действие само по себе не окажет влияния на значение переменной m_dots или значение свойства Dots. Необходимо с помощью мастера ClassWizard связать диалоговое окно с соответствующими переменными-членами, для чего нужно выполнить следующее.

1. Вызовите мастер ClassWizard, оставив диалоговое окно свойств открытым и расположенным поверх всех других окон, и перейдите на вкладку Member Variables.
2. Убедитесь, что выбранным является класс CDierollPropPage и что выделен идентификатор ресурса IDC_DOTS, а затем щелкните на кнопке Add Variable.
3. Введите в поле имени значение m_dots и выберите тип переменной BOOL. В поле Optional Property Name введите значение Dots, как показано на рис. 17.13.
4. Щелкните на кнопке OK, и мастер ClassWizard создаст текст программ, который будет подключать вкладку свойств к указанным переменным-членам, поместив его в функцию CDierollPropPage::OnDataExchange().

Путь, которым выполняется обмен данными, может показаться достаточно извилистым. Когда пользователь открывает окно свойств, значение TRUE или FALSE помещается во временную переменную. Щелчок на флажке опции переключает значение именно этой временной переменной. Когда пользователь делает щелчок на кнопке OK, установленное им значение помещается в CDierollPropPage::m_dots, а также в свойство автоматизации Dots. Это свойство уже было соединено с переменной CDierollCtrl::m_dots, так что карта диспетчера в классе CDierollCtrl обеспечит внесение изменений во все прочие m_dots. Поскольку функция

OnDraw() использует в работе переменную CDierollCtrl::m_dots, внешний вид элемента управления будет изменяться в соответствии с установками, выполненными пользователем на вкладке свойств. То, что одновременно существует две переменных-члена с одинаковыми именами, может вносить некоторую путаницу при разработке вашего первого элемента управления, но со временем вы наберетесь опыта и все станет на свои места.

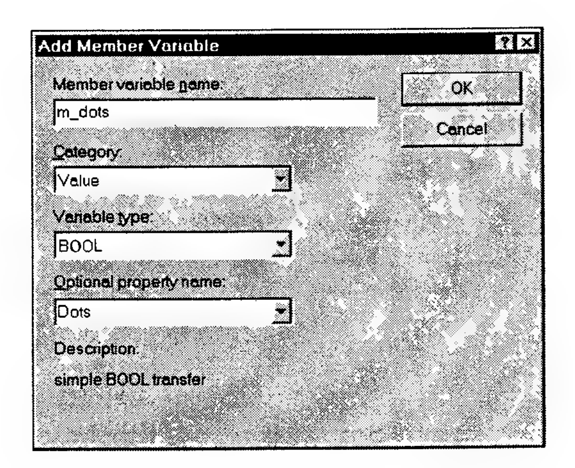


Рис. 17.13. С помощью мастера ClassWizard осуществляется подключение вкладки свойств к свойствам создаваемого элемента управления ActiveX

Задание выполнено. Оттранслируйте приложение и запустите его в тестовом контейнере. Для изменения свойств элемента управления выберите команду Edit⇨Dieroll Control Object⇨Properties. Раскроется созданная вами вкладка свойств, показанная на рис. 17.14. Убедитесь, что элемент управления может отображать значение точками или цифрами в зависимости от установки, выполненной на вкладке свойств. Для этого следует изменить текущую установку опции, щелкнуть на кнопке OK и убедиться, что внешний вид элемента управления изменился соответствующим образом.

Когда наш управляющий элемент отображает свое текущее значение цифрой, пожалуй, имеет смысл отобразить эту цифру шрифтом, пропорции которого лучше соответствуют существующим размерам элемента управления, а также отцентрировать отображаемое значение. Для этого потребуется выполнить относительно несложную модификацию функции OnDraw(), которую я предлагаю вам осуществить собственными силами.

Регулировка цвета

На данный момент созданный нами элемент управления в виде игральной кости всегда будет отображаться как имеющий черные точки на белом фоне. Однако предоставить пользователю возможность управлять выбором цвета изображения очень просто. Все, что нам потребуется, — это свойство для значения цвета объектов переднего плана и еще одно свойство для значения цвета фона. Но данные свойства уже определены как типовые свойства с именами ForeColor и BackColor соответственно.

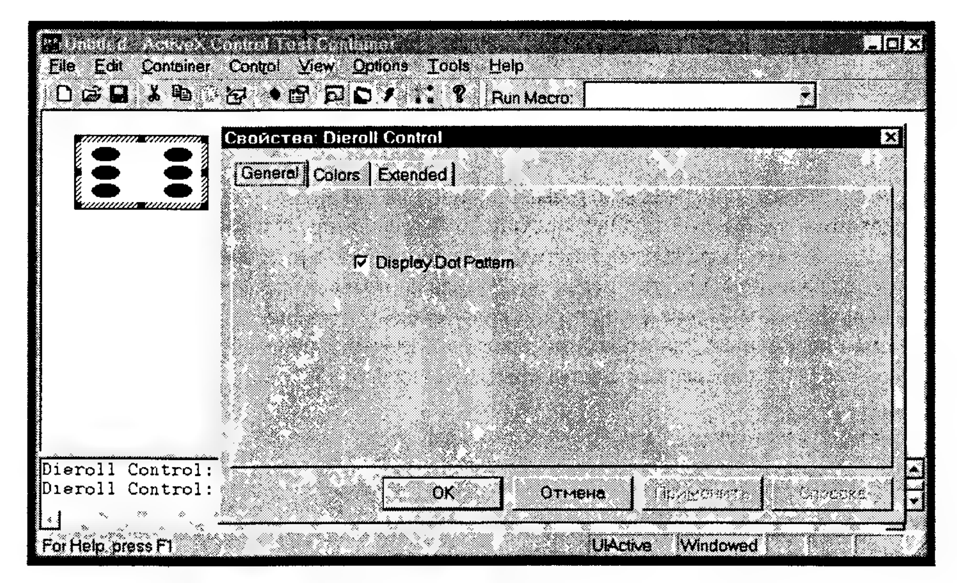


Рис. 17.14. Вновь созданная вкладка свойств, отображенная в окне контейнера тестирования управляющих элементов

Типовые свойства

Ниже приводится полный список типовых свойств, определенных для любого создаваемого вами элемента управления.

- Appearance. Определяет общий вид элемента управления.
- BackColor. Определяет цвет фона элемента управления.
- BorderStyle. Определяет стандартную рамку или отсутствие рамки.
- Caption. Определяет заголовок или текст элемента управления.
- Enabled. Определяет, можно ли использовать данный элемент управления.
- Font. Определяет шрифт, принимаемый по умолчанию для элемента управления.
- ForeColor. Определяет для элемента управления цвет объектов переднего плана.
- Text. Определяет для элемента управления заголовок или текст.
- hWnd. Определяет дескриптор окна элемента управления.

Свойства окружения

Элементы управления могут также иметь доступ к *свойствам окружения*, которые являются свойствами среды, окружающей элемент управления, т.е. свойствами *контейнера*, в который помещен элемент управления. Изменять значения свойств окружения нельзя, однако элемент управления может использовать их для настройки своих собственных свойств. Например, элемент управления может установить собственный цвет фона в соответствии с цветом фона, определенного для контейнера.

Контейнер должен обеспечивать всю необходимую поддержку свойств окружения. Однако везде, где вы используете какое-либо свойство окружения, следует определять и значение по умолчанию, на случай, если контейнер не поддерживает данное свойство окружения. Ниже приведен пример правильного использования в программе свойства окружения, имеющего имя UserMode.

```
BOOL bUserMode;
   if( !GetAmbientProperty( DISPID_AMBIENT_USERMODE,
      VT_BOOL, &bUserMode ) )
   {
      bUserMode = TRUE;
   }
```

В этом фрагменте программы вызывается функция GetAmbientProperty(), на вход которой передаются требуемые диспетчерский идентификатор (dispid) и тип переменной (vartype). Кроме того, передается указатель на переменную, в которую следует поместить запрашиваемое значение. Тип этой переменной должен соответствовать указанному значению vartype. Если функция GetAmbientProperty() возвращает значение FALSE, в данном фрагменте программы переменной bUserMode присваивается значение по умолчанию, равное TRUE.

В файле olectl.h определены следующие значения dispid.

```
DISPID_AMBIENT_BACKCOLOR
DISPID_AMBIENT_DISPLAYNAME
DISPID_AMBIENT_FONT
DISPID_AMBIENT_FORECOLOR
DISPID_AMBIENT_LOCALEID
DISPID_AMBIENT_MESSAGEREFLECT
DISPID_AMBIENT_SCALEUNITS
DISPID_AMBIENT_TEXTALIGN
DISPID_AMBIENT_USERMODE
DISPID_AMBIENT_UIDEAD
DISPID_AMBIENT_SHOWGRABHANDLES
DISPID_AMBIENT_SHOWHATCHING
DISPID_AMBIENT_DISPLAYASDEFAULT
DISPID_AMBIENT_SUPPORTSMNEMONICS
DISPID_AMBIENT_AUTOCLIP
DISPID_AMBIENT_APPEARANCE
```

Не забывайте, что не все контейнеры поддерживают полный список этих свойств. Некоторые из свойств могут не поддерживаться частью приложений-контейнеров, тогда как другие приложения-контейнеры могут поддерживать свойства, отсутствующие в приведенном выше списке.

Параметр vartype может иметь значения, перечисленные в табл. 17.1.

Таблица 17.1. Типы переменных, допустимые для свойств окружения

Значение vartype	Описание	Значение vartype	Описание
VT_BOOL	BOOL	VT_CY	CY
VT_BSTR	CString	VT_COLOR	OLE_COLOR
VT_I2	short	VT_DISPATCH	LPDISPATCH
VT_I4	long	VT_FONT	LPFONTDISP
VT_R4	float		
VT_R8	double		

Помнить о том, какое из значений vartype используется для каждого из dispid, и обеспечивать контроль значения, возвращаемого функцией GetAmbientProperty(), — занятие довольно скучное, поэтому управляющая программа предоставляет некоторый набор методов класса COleControl, обеспечивающих получение значений чаще всего используемых свойств окружения.

- OLE_COLOR AmbientBackColor()
- CString AmbientDisplayName()
- LPFONTDISP AmbientFont() (Не забывайте освобождать шрифт, используя функцию Release().)
- OLE_COLOR AmbientForeColor()
- LCID AmbientLocaleID()
- CString AmbientScaleUnits()
- short AmbientTextAlign() (0 означает стандартный вариант: числа выравниваются вправо, текст — влево; 1 означает выравнивание по левому краю; 2 означает выравнивание по центру; 3 означает выравнивание по правому краю.)
- BOOL AmbientUserMode() (TRUE означает режим пользователя; FALSE означает режим проекта.)
- BOOL AmbientUIDead()
- BOOL AmbientShowHatching()
- BOOL AmbientShowGrabHandles()

Все эти методы обеспечивают разумное значение по умолчанию, если контейнер не поддерживает необходимого свойства окружения.

Подключение свойств BackColor и ForeColor

Для добавления свойств BackColor и ForeColor в создаваемый элемент управления выполните следующие действия.

1. Откройте окно ClassWizard и выберите вкладку Automation.
2. Убедитесь, что выделенным является класс CDierollCtrl, а затем щелкните на кнопке Add Property.
3. В верхнем списке выберите значение BackColor, после чего все остальные поля ввода в окне будут автоматически заполнены, причем текст будет выведен затененным, напоминая вам о том, что для типовых свойств эти значения изменять нельзя (рис. 17.15).
4. Щелкните на кнопке OK, а затем таким же образом добавьте свойство ForeColor. После очередного щелчка на кнопке OK вкладка Automation окна ClassWizard должна приобрести вид, показанный на рис. 17.16. Символ S перед двумя новыми свойствами указывает вам на то, что эти свойства являются типовыми.
5. Для закрытия окна ClassWizard щелкните на кнопке OK.

Определить вкладки свойств для цвета фона и переднего плана почти так же просто, поскольку уже существуют заранее написанные шаблоны вкладок, которыми можно воспользоваться. Найдите в файле DierollCtl.cpp фрагмент текста программ, представленный в листинге 17.14.

Листинг 17.14. Фрагмент файла DierollCtl.cpp — вкладки свойств

```
////////////////////////////////////////////////////////////////////

// Вкладки свойств.

// TODO: если необходимо, добавьте сюда новые вкладки свойств.
```

```
// Не забывайте увеличивать значение счетчика!

BEGIN_PROPPAGEIDS(CDierollCtrl, 1)
   PROPPAGEID(CDierollPropPage::guid)
END_PROPPAGEIDS(CDierollCtrl)
```



Рис. 17.15. Описание типового свойства, созданное мастером `ClassWizard`

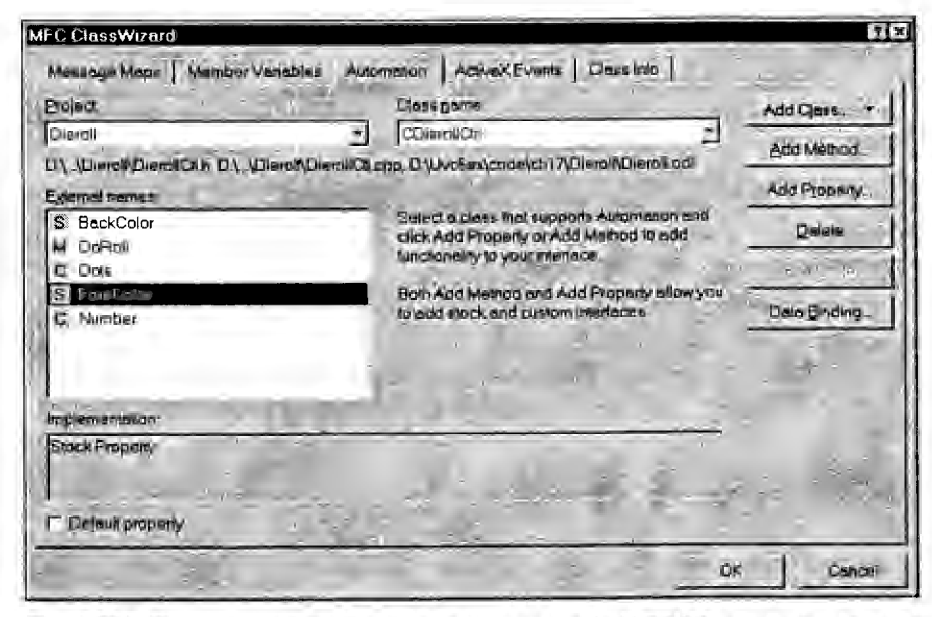


Рис. 17.16. В списке свойств и методов на вкладке **Automation** *типовые свойства помечаются символом S*

Удалите комментарий-напоминание //TODO, исправьте значение счетчика на 2 и добавьте еще одну макрокоманду PROPPAGEID так, как показано в листинге 17.15.

Листинг 17.15. Фрагмент файла DierollCtl.cpp — вкладки свойств

```
///////////////////////////////////////////////////////////////////////////
// Property pages
// Вкладки свойств.

BEGIN_PROPPAGEIDS(CDierollCtrl, 2)
   PROPPAGEID(CDierollPropPage::guid)
   PROPPAGEID(CLSID_CColorPropPage)
END_PROPPAGEIDS(CDierollCtrl)
```

Значение CLSID_CColorPropPage является идентификатором класса вкладки свойств, предназначенной для задания цвета. Теперь, если пользователь выведет на экран окно свойств для данного элемента управления, в нем будут присутствовать уже две вкладки: первая — для установки цвета и вторая — общая вкладка (General), которую мы недавно создали. На вкладке установки цвета будут доступны оба свойства: и Forecolor, и BackColor. Теперь осталось только описать в программе использование цветовых значений, установленных пользователем. Очень скоро все это можно будет увидеть и опробовать непосредственно на экране, но сначала следует откорректировать программу.

Изменения в функции OnDraw()

В функции OnDraw() нашей программы можно определить текущее значение цвета фона с помощью функции GetBackColor(). Раньше эта функция отсутствовала, но она была добавлена мастером ClassWizard, когда в приложении описывались типовые свойства. Карта диспетчера класса CDierollCtrl теперь имеет вид, представленный в листинге 17.16.

Листинг 17.16. Файл DierollCtl.cpp — карта диспетчера

```
BEGIN_DISPATCH_MAP(CDierollCtrl, COleControl)
   //{{AFX_DISPATCH_MAP(CDierollCtrl)
   DISP_PROPERTY_NOTIFY(CDierollCtrl, "Number", m_number, OnNumberChanged, VT_I2)
   DISP_PROPERTY_NOTIFY(CDierollCtrl, "Dots", m_dots, OnDotsChanged, VT_BOOL)
   DISP_STOCKPROP_BACKCOLOR()
   DISP_STOCKPROP_FORECOLOR()
   //}}AFX_DISPATCH_MAP
   DISP_FUNCTION_ID(CDierollCtrl, "AboutBox", DISPID_ABOUTBOX, AboutBox, VT_EMPTY,
VTS_NONE)
END_DISPATCH_MAP()
```

Макрокоманда DISP_STOCKPROP_BACKCOLOR() определена следующим образом.

```
#define DISP_STOCKPROP_BACKCOLOR() \
   DISP_PROPERTY_STOCK(COleControl, "BackColor", \
   DISPID_BACKCOLOR, COleControl::GetBackColor, \
   COleControl::SetBackColor, VT_COLOR)
```

В этом фрагменте программы вызывается другая макрокоманда, DISP_PROPERTY_STOCK(), которая завершает объявление функции GetBackColor() как члена класса CDierollCtrl, наследующего от класса COleControl. По этой причине, несмотря на то что вы не находите описания этой функции, она, тем не менее, доступна для вас. Функция GetBackColor() возвращает значение типа OLE_COLOR, которое преобразуется в тип COLORREF с помощью функции TranslateColor(). Полученное значение COLORREF можно передать в функцию CreateSolidBrush() и затем использовать полученную кисть для рисования фона. Для опре-

деления цвета объектов переднего плана воспользуйтесь функцией GetForeColor(), работать с которой следует аналогичным образом. (Для вывода цифры используйте функцию SetTextColor().) В листинге 17.17 приведен полный текст функции OnDraw() (большая часть текста оператора switch опущена).

Листинг 17.17. Файл DierollCtl.cpp — функция CDierollCtrl::OnDraw()

```
void CDierollCtrl::OnDraw(CDC* pdc, const CRect& rcBounds,
                const CRect& rcInvalid)
{
   COLORREF back = TranslateColor(GetBackColor());
   CBrush backbrush;
   backbrush.CreateSolidBrush(back);
   pdc->FillRect(rcBounds, &backbrush);

   if (!m_dots)
   {
      CString val; //Символьное представление значения типа short.
      val.Format("%i",m_number);
      pdc->SetTextColor(TranslateColor(GetForeColor()));
      pdc->ExtTextOut( 0, 0, ETO_OPAQUE, rcBounds, val, NULL );
   }
   else
   {
      //Точки имеют ширину и высоту в 4 единицы
      // и отстоят на 1 единицу от края.
      int Xunit = rcBounds.Width()/16;
      int Yunit = rcBounds.Height()/16;

      int Top = rcBounds.top;
      int Left = rcBounds.left;

      COLORREF fore = TranslateColor(GetForeColor());
      CBrush forebrush;
      forebrush.CreateSolidBrush(fore);


      CBrush* savebrush = pdc->SelectObject(&forebrush);


      switch(m_number)
         ...
   }
}
```

Еще раз оттранслируйте элемент управления и запустите его на выполнение в тестовом контейнере. Откройте окно свойств, выбрав команду Edit⇨Dieroll Control Object⇨Properties. Как можно видеть из рис. 17.17, вновь созданная страница свойств предоставляет все возможности для определения цвета. Чтобы проверить функционирование новых фрагментов текста программы во всех возможных режимах, несколько раз измените установки для цвета фона и объектов переднего плана, поэкспериментируйте с отображением элемента управления как в цифровом, так и в графическом вариантах.

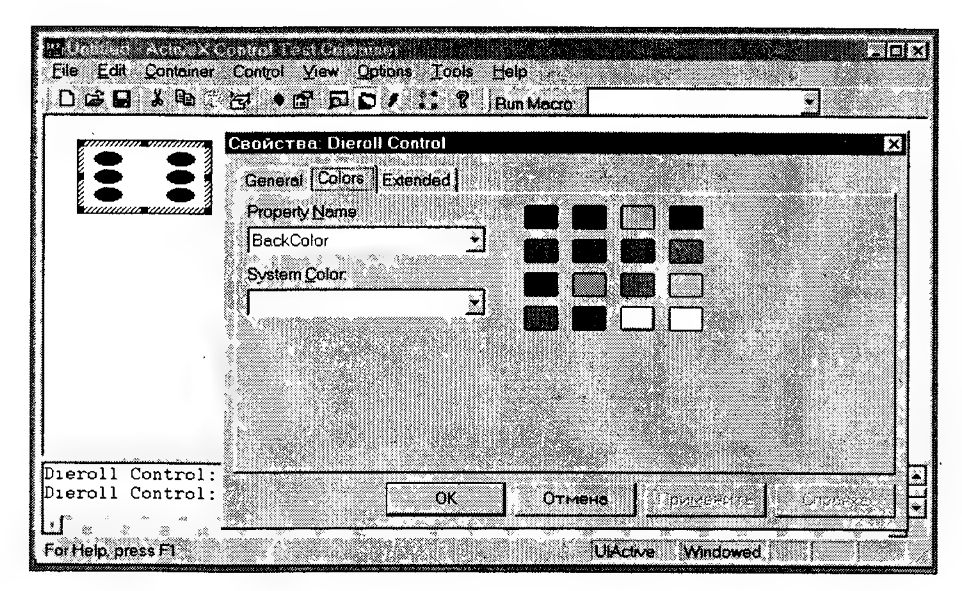


Рис. 17.17. Типовые страницы свойств упрощают работу по предоставлению пользователю возможности контролировать цветовую гамму элемента управления

Имитация броска кости по требованию

Элементы управления ActiveX, так же как и серверы автоматизации, могут предоставлять контейнерам свои функции. Данный элемент управления имитирует бросок игральной кости, когда пользователь выполняет на нем щелчок мышью. Но контейнерному приложению может потребоваться выполнить имитацию броска и без вмешательства пользователя. Для реализации такой возможности необходимо написать метод `DoRoll()` и предоставить контейнеру возможность обращаться к ней.

Запустите мастер `ClassWizard` и выберите вкладку Automation, а затем щелкните на кнопке Add Method. Присвойте новой функции имя `DoRoll`, выберите тип возвращаемого значения `void`. Щелкните на кнопке Edit Code, чтобы добавить новую функцию и получить возможность ввести ее текст, состоящий из следующих строк:

```
void CDierollCtrl::DoRoll()
{
   m_number = Roll();
   InvalidateControl();
}
```

В этом совсем простом фрагменте программы всего лишь имитируется бросок игральной кости и выводится требование перерисовать изображение элемента управления. Все, что касается элементов управления ActiveX, выполняется очень просто!

Теперь можно протестировать созданный элемент — оттранслировать проект, открыть контейнер тестирования, вставить в него элемент управления, а затем выбрать команду Control⇨Invoke Methods. В диалоговом окне Invoke Methods, показанном на рис. 17.18, выберите из раскрывающегося списка `DoRoll (Method)`, а затем щелкните на кнопке Invoke. В ответ элемент управления сымитирует бросок кости.

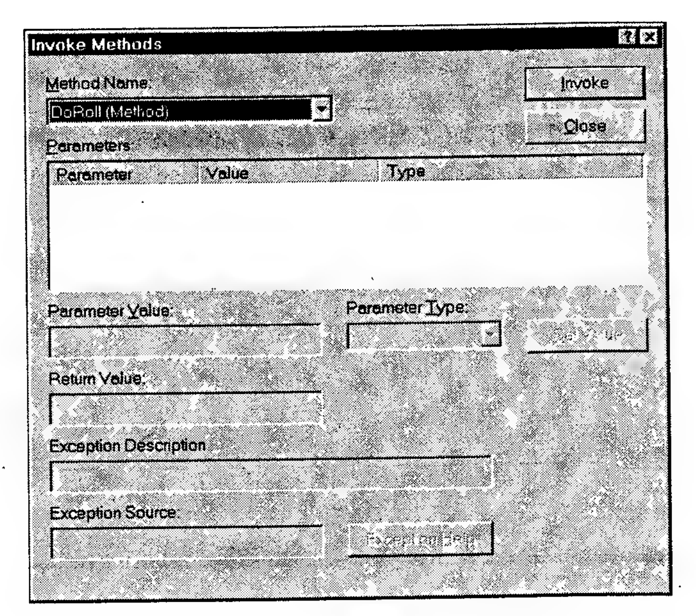


Рис. 17.18. Обратиться к методу элемента управления можно и при помощи диалогового окна Invoke Methods *контейнера тестирования*

Доработка программы

Работа над элементом управления в виде игральной кости кажется вполне завершенной, однако совершенству нет предела.

Активизация и деактивизация элемента

Во многих играх, связанных с игральными костями, вы можете бросить кость только тогда, когда получаете ход. Следовательно, имитация броска кости может быть выполнена только в этот момент и ни в какой другой. Добавив пользовательское свойство с именем `RollAllowed`, вы сможете предоставить контейнеру способ контролировать возможность выполнения броска. Если свойство `RollAllowed` имеет значение `FALSE`, функция `CDierollCtrl::OnLButtonDown()` должна просто возвращать управление, не выполняя имитации броска и перерисовки изображения элемента управления. Возможно, и функция `OnDraw()` должна реагировать на значение свойства `RollAllowed`, отображая элемент как-то иначе (серые точки?), если `RollAllowed` содержит значение `FALSE`. Решать вам. Контейнер, в который внедрен данный элемент управления, будет выполнять установку значений этого свойства так, как и любого другого свойства, предоставляемого сервером автоматизации, сообразуясь с правилами игры, для которой он создан.

Игральная кость с нестандартным числом граней

Зачем ограничивать себя костью, имеющей всегда шесть граней? Если сделать кости, имеющие 4, 8, 12 или даже 30 граней, то не вызовет ли это дополнительный интерес к играм, в которых подобные кости будут использоваться? Вам потребуется изготовить макет одной пары подобных необычных костей, чтобы получить ясное представление об их облике и вне-

сти соответствующие изменения в процедуру рисования в функции CDierollCtrl::OnDraw(). Кроме того, необходимо будет заменить жестко закодированное число 6 в функции Roll() на пользовательское свойство: целое с внешним именем Sides и переменной-членом с именем m_sides. Не забудьте сделать изменения на вкладке свойств, предоставив пользователю возможность определять значение свойства Sides, а также добавить строку в функцию CDierollCtrl::DoPropExchange(), обеспечив сохранение-восстановление свойства Sides и инициализируя его значением 6.



Кость, имвющая две стороны, реально существует — она называется *монета*.

Массив игральных костей

Если вы решите написать программу игры в трик-трак, вам потребуется использовать две игральные кости. Возможный подход состоит во внедрении в контейнер двух экземпляров созданных нами управляющих элементов. Но как синхронизировать их, чтобы они по одному щелчку одновременно выполняли имитацию броска? А почему бы не расширить наш управляющий элемент до *массива* игральных костей? Число костей в этом случае стало бы еще одним пользовательским свойством, а управляющий элемент мог бы осуществлять имитацию броска всех костей одновременно. Флажок RollAllowed в таком случае будет устанавливаться для всех костей одновременно, равно как и свойство Sides. В результате получим возможность использовать две шестигранные или три двенадцатигранные кости, но не две кости, у одной из которых четыре грани, а у другой — двадцать. В последнем случае понадобится превратить свойство Number в массив.



В главе 20 мы будем обсуждать один из способов синхронизации двух или более игральных костей, внедренных в один контейнер, а также некоторые возникающие при этом трудности.

Пустая страница

ЧАСТЬ V

Программирование для Internet

В этой части...

ГЛАВА 18

Windows Sockets, MAPI и Internet

В этой главе...

Использование Windows Sockets

Интерфейс Messaging API — MAPI

Классы для работы с Internet

Классы Internet Server API

Есть несколько способов, с помощью которых программы могут общаться с другими приложениями в Internet. В данной главе описываются основные понятия, применяющиеся при программировании этих способов. В последующих главах некоторые из понятий будут рассмотрены более подробно.

Сеть Internet существовала еще до появления операционной системы Windows. По мере роста она становилась крупнейшей сетью в мире, использующей протокол TCP/IP. Вначале серверами были компьютеры с операционной системой UNIX, и набор соглашений под названием *Berkley sockets* ("Гнезда из Беркли") стал стандартом связи TCP/IP в сети Internet между компьютерами с этой операционной системой. В других операционных системах также был реализован протокол TCP/IP, что способствовало значительному расширению сети Internet. Однако по мере расширения номенклатуры операционных систем началась путаница, вызванная большим количеством различных реализаций TCP/IP, и группа из более чем 20 производителей, объединив свои усилия, создала спецификацию Winsock.

Использование Windows Sockets

Спецификацией Windows Sockets (Winsock) определяется интерфейс динамически загружаемой библиотеки, файл которой, как правило, называется WINSOCK.DLL или WSOCK32.DLL. Функции этой библиотеки реализуются разработчиками. Приложения могут вызывать эти функции и быть уверенными, что имя, смысл аргументов и поведение каждой функции не зависят от конкретной версии установленной библиотеки. Например, библиотеки, включенные в состав Windows 95 и Windows NT, совершенно различны, но 32-битовое приложение, использующее Winsock, может работать без изменений на компьютере как с Windows 95, так и с Windows NT, вызывая функции Winsock в соответствующей библиотеке.

На заметку Спецификация Winsock не ограничивается протоколом TCP/IP. В ней также предусмотрена поддержка протокола IPX/SPX и планируется поддержка других протоколов. Подробнее об этом протоколе можно узнать из самой спецификации Winsock. Отправной точкой может послужить страница Stardust Labs Winsock Resource Page, расположенная по адресу **http://www.stardust.com/wsresource/**.

При программировании с использованием Winsock важным понятием является порт Winsock. Каждому компьютеру в Internet присвоен числовой адрес, называемый IP-адресом, который обычно записывается в виде четырех чисел, разделенных точками. Пусть, например, некоторый компьютер имеет адрес 198.53.145.3. Программы, работающие на нем, хотят с помощью Winsock "общаться" с другими компьютерами. Возникает вопрос: если по адресу 198.53.145.3 приходит запрос, какая программа должна его обрабатывать?

Каждый запрос, приходящий на компьютер, содержит *номер порта* — число от 1024 и выше, указывающее, какой программе предназначается запрос. Некоторые номера портов зарезервированы для стандартного использования. Например, порт 80 традиционно используется серверами Web для получения запросов на документы Web от таких программ-клиентов, как Netscape Navigator.

Работа с Winsock большей частью основана на соединении: две программы устанавливают соединение с помощью *гнезд* (sockets) на каждом конце, а затем передают и принимают данные по этому соединению. Некоторые приложения предпочитают передавать данные без установки соединения, но в этом случае нет гарантии, что данные попадут по назначению. Классическим примером такого приложения является сервер времени, который, не дожидаясь запроса, регулярно посылает информацию о текущем времени всем подключенным к нему компьютерам. Задержка, вызванная установкой соединения, могла бы привести к устареванию посылаемого значения, поэтому в данном случае имеет смысл применить подход, в котором не используется процедура установки соединения.

Winsock в MFC

Поначалу программирование Winsock на Visual C++ сводилось к вызову библиотечных функций API. Многие производители разработали классы, в которых инкапсулированы вызовы этих функций. В Visual C++ 2.1 введены два новых класса — CAsyncSocket и CSocket, унаследованный от CAsyncSocket. Эти классы сами вызывают функции API, включая инициализацию и очистку, о которых легко забыть.

Процесс выполнения программ в Windows является *асинхронным* — в каждый момент времени происходит множество различных событий. В старых версиях Windows, если какая-либо часть приложения зацикливалась или зависала, все приложение (а иногда и вся операционная система) зацикливалось или зависало. Необходимо было всеми средствами этого избегать. В то же время выполнение обращения к гнезду, в частности запроса на чтение информации из другой точки Internet посредством соединения TCP/IP, может занять длительное время. (Функция, ожидающая приема или передачи информации через гнездо, называется *блокирующей*.) Есть три способа решить эту проблему.

1. Выделить функцию, блокирующую выполнение, в отдельный процесс. Этот процесс будет заблокирован, но выполнение остальной части приложения продолжится.
2. Спроектировать функцию так, чтобы она завершалась сразу же после выполнения запроса, а с помощью другой функции периодически проверять (*опрашивать*) гнездо на предмет выполнения запроса.
3. Спроектировать функцию так, чтобы она сразу же возвращала управление, а после выполнения запроса организовать отправку сообщения Windows.

Реализовать первый способ до недавнего времени было невозможно, а использовать второй — неэффективно в среде Windows. Поэтому в большинстве программ, использующих Winsock, применяется третий способ. Этот метод реализован в классе CAsyncSocket. Например, чтобы отослать строку через гнездо с установленной связью в другую точку Internet, необходимо вызвать метод этого гнезда Send(). При вызове Send() не всегда происходит передача данных. Выполняется попытка передачи, и, если гнездо не готово и находится в состоянии ожидания, функция Send() просто завершает свою работу. Когда гнездо освобождается, его окну посылается сообщение. Окно, перехватив это сообщение, передает данные. Реализация такой последовательности операций называется *асинхронным программированием Winsock*.

На заметку Программирование Winsock — непростая тема. Ей были посвящены целые книги. Одна из них, с которой стоит ознакомиться, — *Developing Internet Applications in Visual C++*, изданная Que. Если вы выберете путь низкоуровневого программирования гнезд, начните с построения стандартных приложений.

Класс CAsyncSocket

Класс CAsyncSocket инкапсулирует асинхронные вызовы Winsock. Он содержит набор полезных функций, использующих Winsock API. В табл. 18.1 перечислены методы класса CAsyncSocket и их назначение.

Таблица 18.1. Методы класса CAsyncSocket

Метод	Назначение
Accept	Обрабатывает запрос на соединение, который поступает на принимающее гнездо, заполняя его информацией об адресе
AsyncSelect	Организует посылку сообщения Windows при переходе гнезда в состояние готовности

Метод	Назначение
Attach	Связывает дескриптор гнезда с экземпляром класса CAsyncSocket, чтобы иметь возможность сформировать соединение с другим компьютером
Bind	Ассоциирует адрес с гнездом
Close	Закрывает гнездо
Connect	Подключает гнездо к удаленному адресу и порту
Create	Завершает процесс инициализации, начатый конструктором
Detach	Разъединяет соединение с гнездом с заданным дескриптором
FromHandle	Возвращает указатель на объект класса CAsyncSocket дескриптору, с которым он связан
GetLastError	Возвращает код ошибки гнезда. Вызов этого метода происходит после обнаружения ошибки, чтобы выяснить ее причину
GetPeerName	Определяет адрес IP и номер порта удаленного гнезда, с которым связан вызываемый объект-гнездо, или заполняет этой информацией структуру адреса гнезда
GetSockName	Возвращает адрес IP и номер порта объекта this или заполняет этой информацией структуру адреса гнезда
GetSockOpt	Возвращает текущие параметры гнезда
IOCtl	Устанавливает режим работы гнезда, чаще всего — блокирующий или неблокирующий
Listen	Заставляет гнездо следить за запросами на соединение
OnAccept	Обрабатывает сообщение Windows, которое формируется при приеме гнездом запроса на соединение. Часто переопределяется в производных классах
OnClose	Обрабатывает сообщение Windows, которое формируется при закрытии гнезда. Часто переопределяется в производных классах
OnConnect	Обрабатывает сообщение Windows, которое формируется после установки соединения или после неудачной попытки соединения. Часто переопределяется в производных классах
OnOutOfBandData	Обрабатывает сообщение Windows, которое формируется при появлении в гнезде срочных внеочередных данных, готовых для чтения
OnReceive	Обрабатывает сообщение Windows, которое формируется при появлении данных, которые можно прочесть с помощью Receive(). Часто переопределяется в производных классах
OnSend	Обрабатывает сообщение Windows, которое формируется при готовности гнезда принять данные, посылаемые с помощью Send(). Часто переопределяется в производных классах
Receive	Считывает данные из удаленного гнезда, к которому подключено данное гнездо
ReceiveFrom	Считывает дейтаграмму[2] из удаленного гнезда без установки соединения
Send	Посылает данные удаленному гнезду
SendTo	Посылает дейтаграмму без установки соединения
SetSockOpt	Устанавливает параметры гнезда
ShutDown	Оставляет гнездо открытым, но предотвращает дальнейшие вызовы Send() или Receive()

При использовании класса CAsyncSocket необходимо самостоятельно заполнять структуры адреса гнезда, от чего многие разработчики с радостью отказались бы. В этом случае для программирования гнезд лучше использовать класс CSocket.

[2] Дейтаграмма — пакет в сети передачи данных, передаваемый через сеть независимо от других пакетов без установки логического соединения и квитирования. — *Прим. ред.*

Класс CSocket

Класс CSocket является производным от CAsyncSocket, и поэтому имеет все функции, перечисленные в описании класса CAsyncSocket. В табл. 18.2 перечислены методы, добавленные в класс CSocket, а также виртуальные методы, переопределенные в этом классе.

Таблица 18.2. Методы класса CSocket

Метод	Назначение
Attach	Связывает дескриптор гнезда с экземпляром CAsyncSocket, чтобы он смог установить соединение с другой машиной
Create	Завершает инициализацию после создания конструктором пустого гнезда
FromHandle	Возвращает указатель на объект класса CSocket, связанный с переданным дескриптором гнезда
IsBlocking	Возвращает TRUE, если гнездо в данный момент заблокировано и ожидает каких-то событий
CancelBlockingCall	Отменяет запрос, заблокировавший гнездо
OnMessagePending	Обрабатывает сообщение Windows, генерируемое для других частей приложения в то время, когда гнездо заблокировано. Часто переопределяется в наследуемых классах

Во многих случаях нет необходимости программировать на уровне Winsock API, поскольку классы WinInet, интерфейс ISAPI и элементы управления ActiveX для страниц Web предоставляют программистам все больше разнообразных возможностей. Если вы хотите проанализировать программу, работающую с гнездами, просмотрите приложения Chatter и ChatSrvr, которые входят в комплект поставки Visual C++. Отыщите по любому из этих имен тему в оперативной справке среды разработки.

Во время каждого сеанса работы с Chatter эмулируется сервер пользователя. Программа ChatSrvr является сервером, управляющим передачей данных между несколькими клиентами. Каждый клиент Chatter может посылать сообщения ChatSrvr посредством ввода текста, а ChatSrvr посылает каждое сообщение всем пользователям, подключенным во время текущего сеанса работы. Одновременно обслуживается несколько каналов передачи данных.

Если у вас есть опыт работы с гнездами, достаточно этого краткого обзора для того, чтобы приступить к работе. Если же вы никогда не работали с гнездами, то, возможно, вам и не придется этому учиться. Изучение всех подробностей работы с Winsock необходимо для написания программ типа клиент/сервер, работающих в Internet и не использующих стандартные приложения наподобие e-mail и Web. Если же вы собираетесь оснастить разрабатываемое приложение средствами обращения к электронной почте, Web, FTP и другим популярным информационным службам Internet, для этого не нужно писать программы, напрямую работающие с Winsock. Того же результата можно добиться, воспользовавшись интерфейсами MAPI, ISAPI или классами WinInet.

Интерфейс Messaging API - MAPI

Самая популярная сетевая служба — электронная почта. Можно написать приложение для передачи почтовых сообщений, генерирующее соответствующие команды на уровне Winsock, но проще воспользоваться имеющимся результатом работы других программистов.

Что такое MAPI

MAPI (Messaging API) — это средство, обеспечивающее совместную работу приложений, которым необходимо посылать и принимать сообщения (*приложений, работающих с сообщениями*), и приложений, *умеющих* посылать и принимать сообщения (*служб обработки сообщений*), разработанное для облегчения жизни разработчикам программного обеспечения. На рис. 18.1 показана область применения MAPI. Необходимо отметить, что термин *сообщение* в действительности означает намного больше, чем просто электронную почту: средства MAPI могут обеспечить пересылку не только электронных, но и факсимильных, а также звуковых сообщений. Если в приложении используется MAPI, то установленные пользователем средства обработки сообщений, такие как клиенты электронной почты, будут выполнять всю работу по пересылке генерируемых этим приложением сообщений.

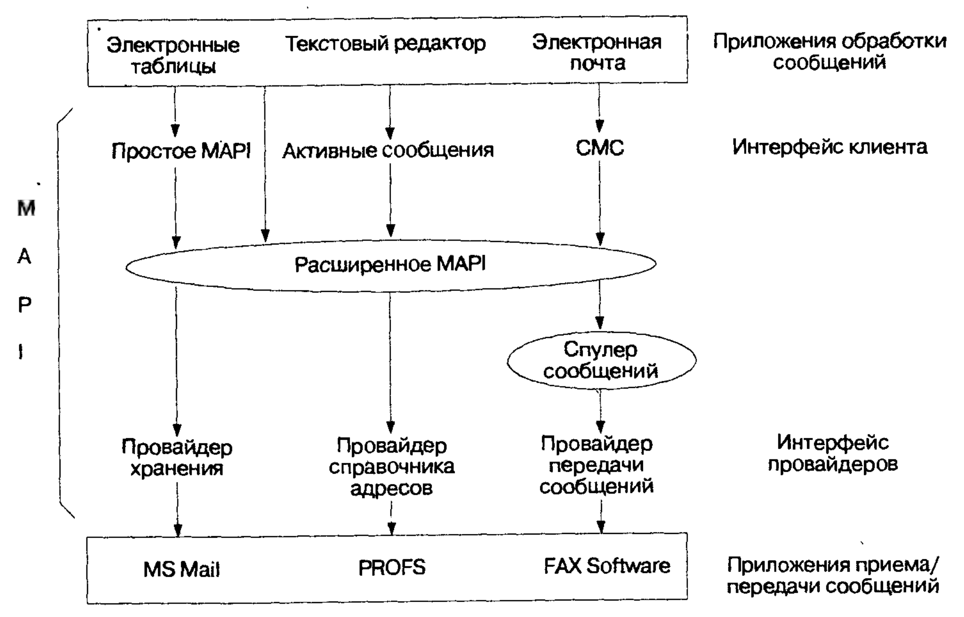


Рис. 18.1. MAPI объединяет приложения, которым нужна обработка сообщений, и приложения, осуществляющие эту обработку

В разных приложениях сообщения могут использоваться по-разному.

- Некоторые приложения могут посылать сообщения, но это не является их основной задачей. Например, текстовый процессор предназначен для ввода и форматирования текста, последующей его распечатки или сохранения. Хорошо если текстовый процессор может к тому же послать текст в виде сообщения, но это уже не так важно. О приложениях такого типа принято говорить, что они *поддерживают сообщения*, и в них обычно используются лишь некоторые функции MAPI.
- Некоторые приложения вполне справляются с возложенными на них задачами и без функций обмена сообщениями, но в среде с возможностью такого обмена их ценность значительно возрастает. Например, программа — организатор личного расписания может работать со списком задач отдельного пользователя, но, если имеется возможность работы с сообщениями, становится доступным набор функций для групповой

работы или контактов с клиентами. Подобные приложения называются *использующими сообщения*, и в них обычно задействованы многие, но не все возможности MAPI.

- Наконец, некоторые приложения предназначены для работы с сообщениями. Без поддержки сообщений они вообще не имеют смысла. О таких приложениях говорят, что они *основаны на сообщениях*, и в них используются все функции MAPI.

Требования сертификата Win95

Для производителей причиной номер один создания приложений, поддерживающих сообщения, является удовлетворение требованиям *сертификата Windows 95*. Чтобы получить право на такой сертификат, в меню File приложения должен быть пункт Send, а соответствующая команда должна обеспечить пересылку документа с помощью MAPI (исключение сделано для приложений, в которых нет документов).

О добавлении в приложение такой функции лучше подумать при планировании настройки AppWizard на этапе создания заготовки. Если вы желаете, чтобы приложение удовлетворяло этой части требований сертификата, необходимо проделать следующую работу.

1. На этапе 4 настройки AppWizard установите флажок MAPI (Messaging API).

И это все! Требуемый пункт меню добавлен, и созданы карты сообщений и функции, обрабатывающие эту команду и вызывающие функции, которые посылают документ посредством MAPI, используя вашу функцию Serialize(). На рис. 18.2 показано окно приложения MAPIDemo, которое является просто заготовкой, сформированной AppWizard.

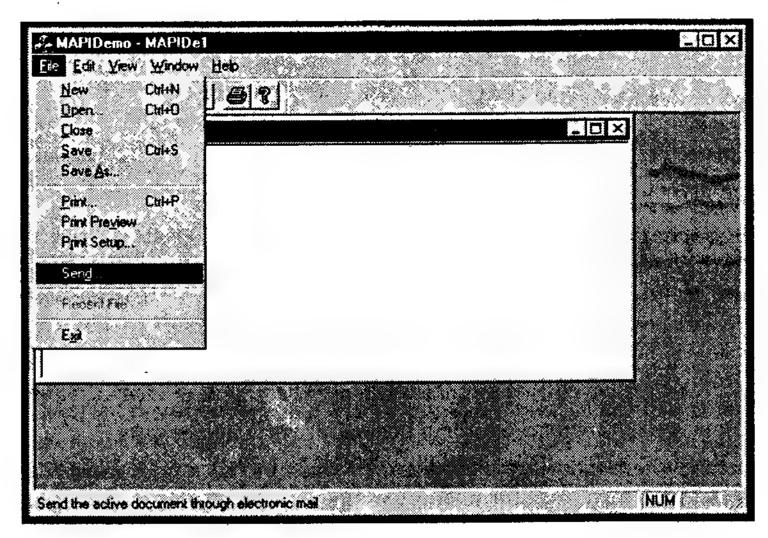


Рис. 18.2. AppWizard добавляет в меню File пункт Send, а также программу обработки этой команды

В этом приложении нет ничего, кроме текста, созданного AppWizard, а в меню File появился пункт Send. При выборе этого пункта меню запускается почтовый клиент MAPI для посылки сообщения. Посылаемое сообщение содержит текущий документ, и вам остается заполнить поля адреса и темы сообщения и дописать любой текст, который вы хотите послать с этим документом.



Если пункт Send не появился в меню, проверьте, установлено ли клиентское программное обеспечение MAPI. К числу клиентов MAPI, доступных большинству пользователей, принадлежит Microsoft Exchange. Если на компьютере не зарегистрирован клиент MAPI, функция OnUpdateFileSendMail() удаляет из меню пункт Send.

Если вы не указали поддержку MAPI при построении приложения с помощью AppWizard, добавьте пункт меню Send вручную.

1. Добавьте в меню File пункт Send. В качестве идентификатора ресурса используйте ID_FILE_SEND_MAIL. Текст будет добавлен автоматически.
2. Добавьте в карту сообщений документа за пределами комментариев //AFX следующие строки.

```
ON_COMMAND(ID_FILE_SEND_MAIL, OnFileSendMail)
ON_UPDATE_COMMAND_UI(ID_FILE_SEND_MAIL, OnUpdateFileSendMail)
```

Добавить поддержку почты в приложение вручную не намного сложнее, чем настроить AppWizard.

Расширенное использование MAPI

Если вы хотите от MAPI больше, чем просто удовлетворять требованиям сертификата, все становится сложнее. Существует четыре интерфейса клиентов MAPI.

- *Simple MAPI* (простой MAPI) — устаревшая версия API, которую не рекомендуется использовать в новых приложениях.
- *Common Messaging Calls, CMC* (единая система обработки сообщений) — простой API для приложений, поддерживающих и использующих сообщения.
- *Extended MAPI* (расширенный MAPI) — полный API для приложений, основанный на сообщениях.
- *OLE Messaging* (обработка сообщений OLE) — интерфейс с несколько меньшим, чем в расширенном API, набором возможностей, но идеальный для Visual C++.

Интерфейс CMC

В состав интерфейса CMC входит всего десять функций. Их несложно запомнить, а возможностей CMC вполне хватает для выполнения достаточно широкого круга задач. В протокол включены следующие функции:

- cmc_logon() — устанавливает связь с почтовым сервером и идентифицирует пользователя;
- cmc_logoff() — завершает связь с почтовым сервером;
- cmc_send() — посылает сообщение;
- cmc_send_documents() — посылает один или более файлов;
- cmc_list() — получает список сообщений в почтовом ящике;
- cmc_read() — считывает сообщение из почтового ящика;
- cmc_act_on() — сохраняет или удаляет сообщение;
- cmc_look_up() — находит имена и адреса;
- cmc_query_configuration() — возвращает сводку об используемом почтовом сервере;
- cmc_free() — освобождает память, распределенную другими функциями.

В файле заголовка XCVC.H описано несколько структур, предназначенных для хранения информации, передаваемой этим функциям. Например, в следующей структуре хранится информация о получателе.

```
/*RECIPIENT*/
typedef struct {
   CMC_string          name;
   CMC_enum            name_type;
   CMC_string          address;
   CMC_enum            role;
   CMC_flags           recip_flags;
   CMC_extension FAR   *recip_extensions;
} CMC_recipient;
```

Можно заполнить эту структуру данными об имени и адресе получателя сообщения с помощью стандартного диалогового окна, а можно указать эти значения непосредственно в тексте программы.

```
CMC_recipient recipient = {
   "Kate Gregory",
   CMC_TYPE_INDIVIDUAL,
   "SMTP:kate@gregcons.com",
   CMC_ROLE_TO,
   CMC_RECIP_LAST_ELEMENT,
   NULL };
```

В полях типа (name_type), роли (role) и флагов (recip_flags) используются следующие предопределенные значения (листинг 18.1)

Листинг 18.1. Константы (фрагмент файла \MSDev\Include\XCMC.H)

```
/* типы имен */
#define CMC_TYPE_UNKNOWN              ((CMC_enum) 0)
#define CMC_TYPE_INDIVIDUAL           ((CMC_enum) 1)
#define CMC_TYPE_GROUP                ((CMC_enum) 2)

/* роли */
#define CMC_ROLE_TO                   ((CMC_enum) 0)
#define CMC_ROLE_CC                   ((CMC_enum) 1)
#define CMC_ROLE_BCC                  ((CMC_enum) 2)
#define CMC_ROLE_ORIGINATOR           ((CMC_enum) 3)
#define CMC_ROLE_AUTHORIZING_USER     ((CMC_enum) 4)

/* флаги получателя */
#define CMC_RECIP_IGNORE              ((CMC_flags) 1)
#define CMC_RECIP_LIST_TRUNCATED      ((CMC_flags) 2)
#define CMC_RECIP_LAST_ELEMENT        ((CMC_flags) 0x80000000)
```

Существует также структура сообщения, которая заполняется теми же способами — программно или с помощью диалогового окна, предлагаемого пользователю для ввода реквизитов сообщения. Эта структура включает указатель на уже заполненную структуру получателя (типа CMC_recipient). Затем процесс завершается вызовами функций cmc_logon(), cmc_send() и cmc_logoff().

Интерфейс Extended MAPI

Интерфейс Extended MAPI базируется на COM — модели объектов-компонентов OLE (OLE Component Object Model). Сообщения, получатели и многие другие понятия определены не как структуры С, а как объекты. Объектов в Extended MAPI намного больше, чем структур в CMC. Доступ к этим объектам производится с помощью интерфейсов OLE (ActiveX). Объекты экспортируют свойства, методы и события. Эти понятия рассмотрены в главе 14.

Интерфейс OLE Messaging

Если вы поняли сущность автоматизации ActiveX (описанной в главе 16), вам несложно будет разобраться с интерфейсом OLE Messaging. Однако для этого приложение должно быть клиентом автоматизации ActiveX, а построение подобных приложений-клиентов выходит за рамки данной главы. Одним из вариантов использования этого интерфейса является программирование на Visual Basic и разработка для таких приложений, как Excel, сценариев на этом языке. Подобная программа создавала бы необходимые объекты, а затем устанавливала их экспортируемые свойства (например, строку тематического заголовка объекта сообщения) и вызывала их предоставляемые методы (например, метод `Send()` объекта сообщения).

К объектам, используемым в OLE Messaging, относятся следующие:

- сеанс работы (session);
- сообщение (message);
- получатель (recipient);
- присоединяемый файл (attachment).

Подробный справочник по этим объектам, их свойствам и методам находится в файлах справки среды Visual Studio. Обратитесь к файлам справки по следующим темам: *SDK*, *Win32 SDK*, *Win32 Messaging (MAPI)* и *OLE Messaging Library*.

Классы для работы с Internet

В MFC 4.2 был введен набор новых классов, использование которых освобождает от необходимости изучать программирование Winsock для доступа к стандартным клиентским службам Internet. На рис. 18.3 показана связь между этими классами. В совокупности они называются классами WinInet, и в их число входят следующие классы.

- `CInternetSession`
- `CInternetConnection`
- `CInternetFile`
- `CHttpConnection`
- `CHttpFile`
- `CGopherFile`
- `CFtpConnection`
- `CGopherConnection`
- `CFileFind`
- `CFtpFileFind`
- `CGopherFileFind`
- `CGopherLocator`
- `CInternetException`



С помощью этих классов можно писать *клиентские* программы Internet, с которыми работает непосредственно пользователь. Если вы хотите писать *серверные* приложения, взаимодействующие с клиентскими приложениями, вам понадобятся классы ISAPI, обсуждаемые в следующем разделе.

Программа начинает сеанс работы с создания экземпляра класса `CInternetSession`. Затем, если у вас есть адрес ресурса (URL) Gopher, FTP или Web (HTTP), вы можете получить этот ресурс в виде доступного только для чтения объекта класса `CInternetFile` с помощью вызова

функции CInternetSession::OpenURL() только что созданного экземпляра. Затем приложение может считывать содержимое этого файла с помощью функций класса CStdioFile и обрабатывать данные.

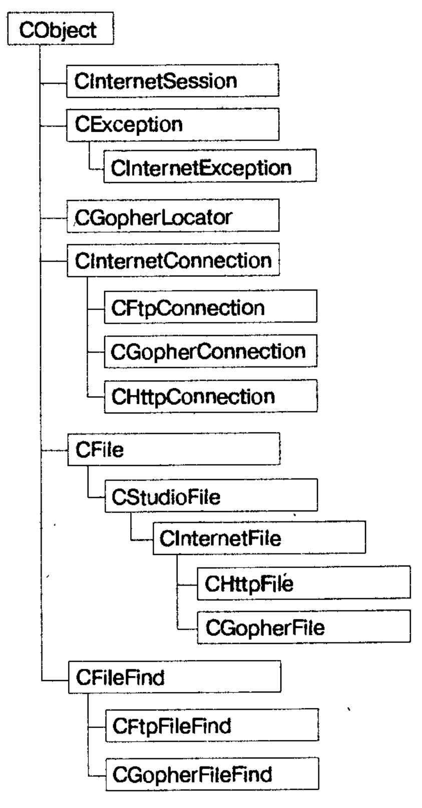


Рис. 18.3. Классы WinInet, облегчающие написание клиентских программ Internet

Если у вас нет адреса ресурса или вы не хотите получить файл, доступный только для чтения, после начала сеанса работы действуйте иначе. Сначала устанавливается связь по указанному протоколу с помощью вызова функций CetFtpConnection(), GetGoopherConnection() и GetHttpConnection() класса CInternetSession, возвращающих соответствующий объект соединения. Затем происходит вызов функции объекта соединения, причем CFtpConnection::OpenFile() возвращает объект класса CInternetFile, CGopherConnection::OpenFile() возвращает объект класса CGopherFile, а CHttpConnection::OpenFile возвращает объект класса CHttpFile. С помощью класса CFileFind и производных от него можно отыскать необходимый файл.

В главе 19 анализируется пример клиентской программы, в которой классы WinInet используются для открытия сеанса работы с Internet и получения информации.

На заметку Несмотря на то что электронная почта является стандартным приложением Internet, среди классов WinInet нет классов, работающих с электронной почтой, поскольку электронная почта обрабатывается с помощью MAPI. Также ни в классах WinInet, ни где-либо еще нет поддержки службы новостей Usenet.

Классы Internet Server API

Интерфейс ISAPI (Internet Server API) используется для расширения возможностей сервера HTTP (World Wide Web). Разработчики ISAPI создают *расширения* и *фильтры*. Расширения — это модули динамически загружаемых библиотек (DLL), вызываемые пользователем на страницах Web практически так же, как и приложения CGI. Фильтры — это модули DLL, выполняемые на сервере и просматривающие или модифицирующие данные, которые поступают на сервер или передаются сервером. Например, с помощью фильтра можно перенаправить запросы на какой-либо файл в другое место.

На заметку

Чтобы написанные вами расширения и фильтры ISAPI работали, включающие их страницы Web должны располагаться на ISAPI-совместимом сервере, например `Microsoft IIS Server`. Нужно иметь разрешение на установку модулей DLL на сервер, а в случае с фильтрами ISAPI вы должны иметь возможность изменять реестр сервера. Если еаши страницы Web хранятся на машине, администрирование которой осуществляет ваш провайдер услуг Internet (Internet Service Provider — ISP), расширить возможности страниц Web с помощью ISAPI будет, скорее всего, невозможно. Вероятно, вам захочется перенести свои страницы на отдельный сервер, чтобы пользоваться ISAPI, но это потребует значительных расходов. Хорошей комбинацией для сервера является мощный компьютер на базе процессора Intel под управлением Windows NT Server 4.0 и Microsoft IIS. Перед тем как приступать к проекту, использующему ISAPI, проанализируйте ограничения, накладываемые существующим сервером Web.

Одним из главных преимуществ элементов управления ActiveX для Internet (см. главу 20) является то, что для их реализации не нужен доступ к серверу.

Пять классов MFC ISAPI образуют оболочку API, которая облегчает его использование. К ним относятся следующие классы.

- `CHttpServer`
- `CHttpFilter`
- `CHttpServerContext`
- `CHttpFilterContext`
- `CHtmlStream`

В приложении должен быть класс сервера или фильтра (или оба), производный от `CHttpServer` или `CFilterServer`. Они похожи на классы в обычном приложении, унаследованном от `CWinApp`. Существует только один экземпляр каждого класса в каждом модуле DLL, а взаимодействие сервера с клиентом осуществляется с помощью его собственного экземпляра соответствующего класса контекста (`CHttpServerContext` или `CHttpFilterContext`). В библиотеке может быть как сервер, так и фильтр, но не более одного экземпляра каждого класса. `CHtmlStream` — вспомогательный класс, описывающий поток HTML, который посылается сервером клиенту.

Мастер расширений ISAPI (`ISAPI Extension Wizard`) — это `AppWizard`, упрощающий создание расширений и фильтров. Для того чтобы воспользоваться этим мастером, необходимо, как всегда, выбрать File⇨New, а затем переключиться на вкладку Projects. После этого необходимо в списке, расположенном в левой части окна, выбрать `ISAPI Extension Wizard` (рис. 18.4), заполнить поля названия и папки проекта и щелкнуть на кнопке OK.

Создание расширения сервера происходит за один шаг, являющийся одновременно первым этапом создания фильтра. Соответствующее диалоговое окно мастера показано на рис. 18.5. Имя и описание фильтра или расширения основаны на выбранном имени проекта.

При создании фильтра активизируется кнопка Next, и можно перейти ко второму этапу настройки, показанному на рис. 18.6. Исходя из объема списка параметров можно сделать вывод о мощности фильтра ISAPI. С его помощью имеется возможность следить за всеми входящими и исходящими запросами и данными, аутентификацией пользователей, вести журнал пересылок и выполнять другие действия.

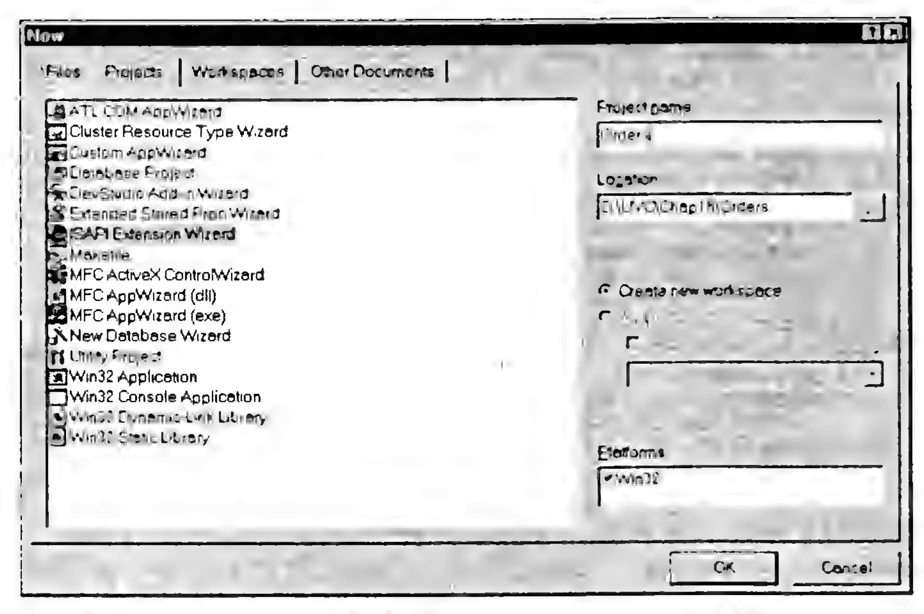


Рис. 18.4. Мастер расширений ISAPI — это разновидность AppWizard

Рис. 18.5. Первый этап настройки мастера расширений ISAPI — присвоение имен компонентов создаваемой библиотеки

Перед созданием файлов AppWizard выводит на экран заключительное окно подтверждения. При одновременном создании сервера и фильтра автоматически создаются 11 файлов, включая файлы реализации и файлы заголовков классов, производных от CHttpServer и CHttpFilter.

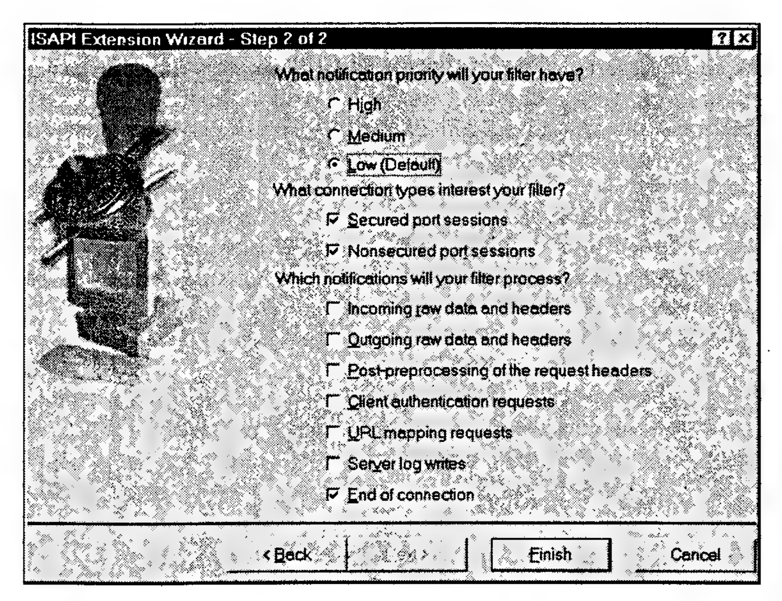


Рис. 18.6. Второй этап настройки мастера расширений ISAPI — установка параметров фильтра

На основе этой заготовки написать фильтр довольно просто. Для каждого сообщения, извещение о котором было запрошено на втором этапе настройки, в заготовку включен прототип обрабатывающей его функции. Например, в классе фильтра имеется функция OnEndOfNetSession(), вызываемая после окончания сеанса работы клиента с данным сервером. В эту функцию добавляется текст программы, который должен обеспечить учет, слежение или любую другую реакцию на это событие. После завершения создания фильтра необходимо вручную отредактировать реестр сервера, чтобы сервер мог запустить выполнение разработанного модуля DLL.

При написании расширения в библиотеку добавляется одна или более функций. Каждой из них передается указатель на объект класса CHttpContext, с помощью которого можно получить такую информацию, как адрес IP пользователя. Если функция вызывается из экранной формы HTML, ей будут переданы дополнительные параметры, такие как значения других полей формы.

От конкретного приложения зависит, что именно делает данная функция. При реализации электронной системы заказов используемые функции будут длинными и сложными. Расширения другого назначения могут быть проще.

После написания функции модуль DLL помещается в папку с модулями, выполняемыми на сервере (обычно — в папку с программами CGI), а страницы Web модифицируются так, чтобы в них содержались ссылки на этот модуль. Например:

```
Теперь вы можете <A HREF=http://www.company.com/exec/orders.dll> сделать заказ</A>
прямо в сети!
```

Более полная информация о программировании с помощью ISAPI содержится в изданной Que книге *Special Edition Using ISAPI* (Специальное издание: использование ISAPI). В ней рассказывается, как с помощью ISAPI сделать страницы Web динамическими и интерактивными, написать фильтры и расширения, а также освещаются специальные темы, включая отладку приложений ISAPI и написание приложений, содержащих несколько параллельно выполняемых задач.

Добавление возможности работать с Internet в приложения — увлекательное направление работы. В этой области программистов ожидает еще много проблем, но в результате будут созданы программные продукты с разнообразными возможностями, упрощающие жизнь каждому человеку, чей компьютер подключен к Internet. Всего год назад написание программ для Internet обозначало погружение в программирование Winsock, запоминание портов TCP/IP и чтение RFC. Новые классы WinInet и ISAPI, а также совершенствование прежних средств поддержки MAPI означают, что сегодня с помощью нескольких строк программы или установленного флажка в окне настройки `AppWizard` можно значительно расширить возможности приложений.

ГЛАВА 19

Использование классов WinInet при программировании для Internet

В этой главе...

Разработка программы опроса Internet

Создание диалогового окна Query

Опрос серверов HTTP

Опрос серверов FTP

Опрос серверов Gopher

Отправка запроса Finger с помощью Gopher

Над чем еще стоит поработать

Разработка программы опроса Internet

В главе 18 были представлены классы WinInet, на основе которых можно создавать клиентские приложения Internet на относительно простом уровне. В данной главе разрабатывается приложение Internet, иллюстрирующее использование классов WinInet. Это приложение также выполняет полезную функцию: с его помощью можно получить информацию о присутствии в Internet какой-либо компании или организации. Для этого не придется изучать работу с гнездами или вдаваться в детали протоколов Internet.

Представьте, что у вас есть чей-то электронный адрес (например, `kate@gregcons.com`) и вы хотите узнать больше о домене (в этом примере — о `gregcons.com`). Или, возможно, вы придумали хорошее имя домена и хотите знать, использовано ли оно. Приложение, о котором пойдет речь в данной главе (`Query`), пытается установить соединение с `gregcons.com` (или с `greatidea.org`, или с любым другим указанным доменом) несколькими различными способами и сообщает о результатах этих попыток.

Пользовательский интерфейс рассматриваемого приложения прост. Единственной информацией, вводимой пользователем, является имя опрашиваемого домена, и сохранять эту информацию в документе нет необходимости. Можно создать пункт меню Query (Запросить), после выбора которого на экран выводится диалоговое окно, предназначенное для ввода адреса, но лучше построить приложение на основе диалогового окна, имеющего кнопку Query.

У приложений на основе диалогового окна, как описано в соответствующем разделе главы 1, нет ни документов, ни строки меню. На экране постоянно отображается диалоговое окно, закрытие которого приводит к закрытию приложения. Само диалоговое окно этого приложения, как и любого другого, создается с помощью Visual Studio.

Чтобы создать заготовку приложения, выберите в меню Visual Studio File⇨New и щелкните на вкладке Project. Выделите `MFC AppWizard (exe)`, присвойте приложению имя `Query`, а на первом этапе настройки выберите приложение на основе диалогового окна (переключатель Dialog based) (рис. 19.1). Затем щелкните на кнопке Next, чтобы перейти ко второму этапу настройки `AppWizard`.

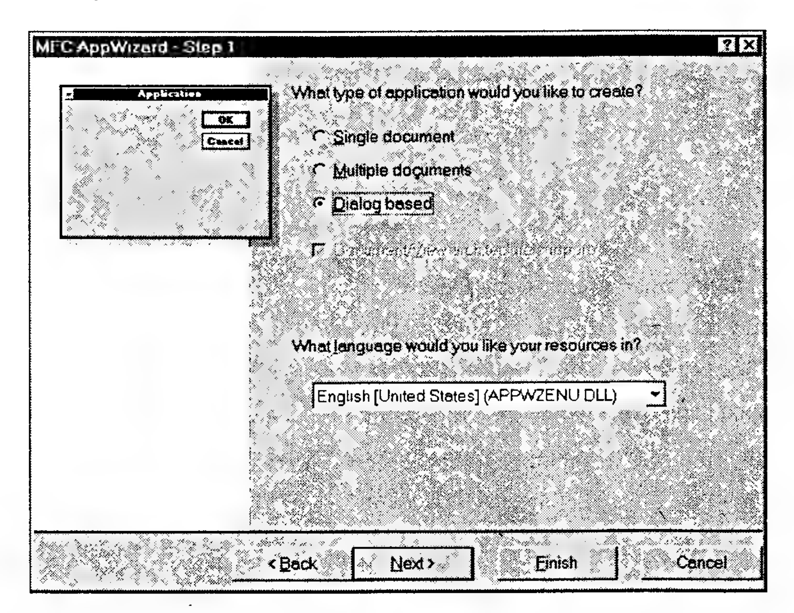


Рис. 19.1. Выбор приложения на основе диалогового окна

Здесь закажите наличие окна About (флажок About box) и объемный дизайн элементов управления (флажок 3D controls). Флажки остальных опций настройки — контекстной справки, автоматизации, поддержки элементов ActiveX и поддержки Windows Sockets — должны быть сброшены. (В этом приложении функции работы с гнездами не будут вызываться непосредственно.) Присвойте диалоговому окну приложения осмысленное название. Параметры настройки AppWizard показаны на рис. 19.2. Затем с помощью кнопки Next перейдите к следующему этапу настройки.

Оставшаяся часть процесса создания приложения с помощью AppWizard вам уже знакома: необходимы комментарии, библиотеки MFC должны быть разделяемыми модулями DLL, изменять имена классов, предложенные AppWizard, не нужно. После завершения настройки AppWizard можно приступать к разработке нетривиальной части приложения Query.

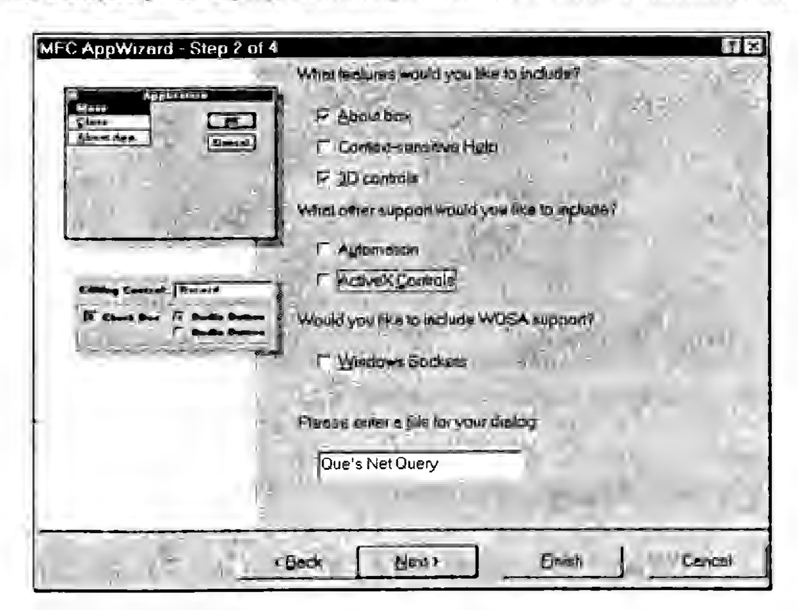


Рис. 19.2. Этому приложению не требуется система оперативной справки, поддержка автоматизации, элементов управления ActiveX и интерфейса Winsock

Создание диалогового окна Query

Сначала AppWizard создает пустое диалоговое окно (рис. 19.3). Для его редактирования необходимо переключиться в режим просмотра ресурсов, раскрыть ветвь Query resources, а в ней — Dialog и дважды щелкнуть на идентификаторе ресурса IDD_QUERY_DIALOG. В дальнейшем это диалоговое окно станет интерфейсом приложения Query.



Если вы еще не работали с диалоговыми окнами, прочтите главу 2.

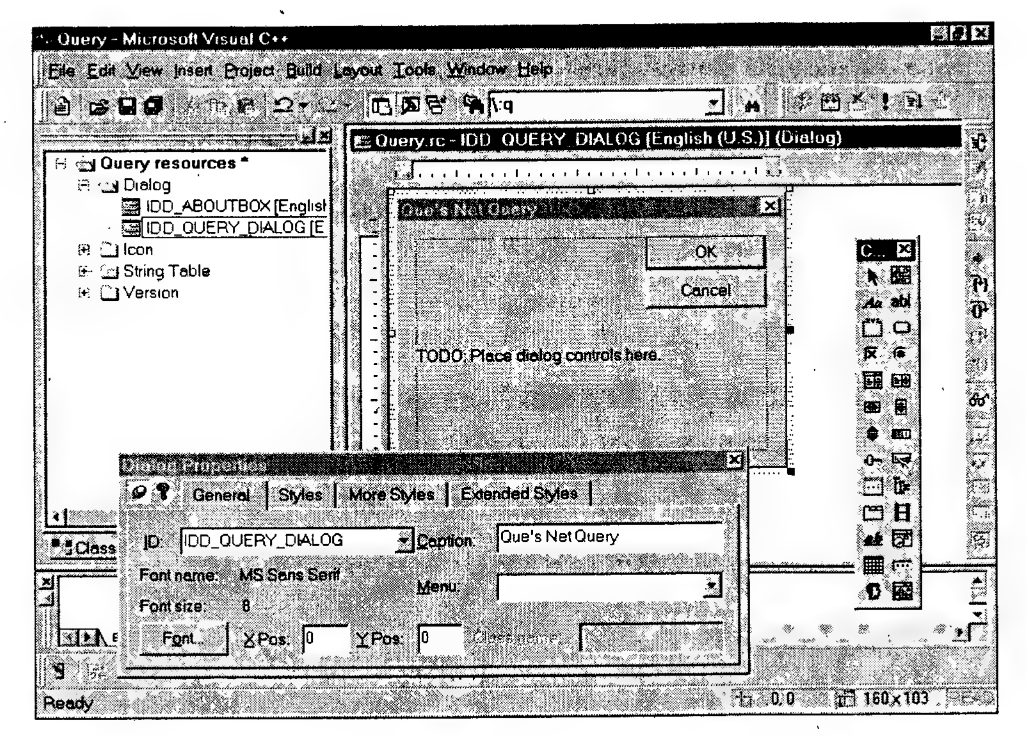


Рис. 19.3. Пустое диалоговое окно, генерируемое `AppWizard`

1. Замените надпись на кнопке `OK` надписью `Query`.
2. Замените надпись на кнопке `Cancel` надписью `Close`.
3. Удалите статический текст `TODO`.
4. Перетащите правый край диалогового окна так, чтобы его ширина была не менее 300 пикселей. (Размер выбранного в данный момент объекта отображается в правом нижнем углу экрана.)
5. В верхней части диалогового окна добавьте текстовое поле с идентификатором ресурса `IDC_HOST`. Его ширина должна быть максимально возможной.
6. Добавьте статический текст (этикетку) `Site name` слева от текстового поля.
7. Перетащите нижнюю границу окна так, чтобы его размер был не менее 150 пикселей.
8. Добавьте еще одно текстовое поле, заполняющее все оставшееся пространство в нижней части окна.
9. Присвойте ему идентификатор ресурса `IDC_OUT`.
10. Щелкните на вкладке Styles окна Properties и установите флажки Multiline, Horizontal scroll. Vertical scroll, Border и Read-only. Проверьте, чтобы все остальные флажки были сброшены.

В законченном виде диалоговое окно и свойства большого текстового поля, установленные на вкладке Styles, должны выглядеть так, как показано на рис. 19.4.

В ответ на щелчок на кнопке Query приложение должно каким-то образом опросить указанный адрес. Последний шаг в создании интерфейса — связывание кнопки Query с текстом программы с помощью `ClassWizard`. Для организации связи проделайте следующее.

Рис. 19.4. Пользовательский интерфейс приложения Query, построенный в виде единственного диалогового окна

1. Выведите на экран окно ClassWizard с помощью команды View⇨ClassWizard.
2. Команду, вызываемую после щелчка на кнопке Query, могут перехватывать три класса, но логично будет выбрать `CQueryDlg`, поскольку в этом классе известно указанное имя сервера. Убедитесь, что в раскрывающемся списке Class name выбран класс `CQueryDlg`.
3. Выделите в левом списке `IDOK` (идентификатор ресурса кнопки OK при изменении надписи остался прежним), а в правом — `BN_CLICKED`.
4. Щелкните на кнопке Add Function, чтобы добавить функцию, вызываемую после щелчка на кнопке Query.
5. `ClassWizard` предлагает имя функции `OnOK`. Введите вместо него `OnQuery` (рис. 19.5) и щелкните на OK.

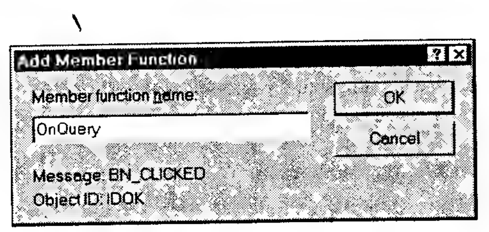


Рис. 19.5. Добавление функции обработки щелчка мышью на кнопке Query (несмотря на изменение надписи на кнопке, ее идентификатором по-прежнему является IDOK)

6. Щелкните на вкладке **Member Variables**, чтобы подготовиться к связыванию текстовых полей диалогового окна с переменными класса этого окна.

7. Выделите IDC_HOST и щелкните на кнопке **Add Variable**. Как показано на рис. 19.6, этот элемент управления связывается с переменной m_host типа CString — членом класса диалогового окна.

8. Свяжите IDC_OUT с переменной m_out, также типа CString.

Закройте окно ClassWizard, щелкнув на кнопке OK. Теперь остается только написать функцию CQueryDlg::OnQuery(), в которой будет использоваться значение переменной m_host для вывода строк текста в окно m_out.

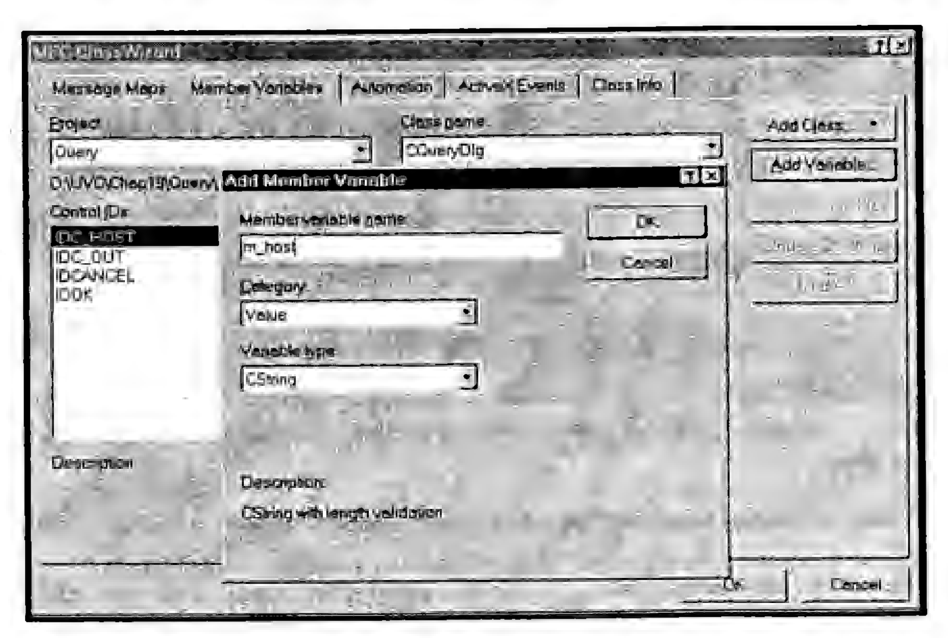


Рис. 19.6. Связывание IDC_HOST с CQueryDlg::m_host

Опрос серверов HTTP

При поиске домена Internet необходимо начинать с соединения HTTP, поскольку на многих серверах есть страницы Web. Самый простой способ осуществить соединение по протоколу HTTP — вызвать метод CInternetSession::OpenURL() (класс CInternetSession входит в состав WinInet). Он возвращает файл, первые несколько строк которого можно вывести в поле m_out. Прежде всего необходимо добавить в начале файла QueryDlg.cpp следующую строку:

```
#include "afxinet.h"
```

После этого можно пользоваться классами WinInet. Поскольку в приложении будет проверяться более чем один адрес, создайте для этого метод CQueryDlg::TryURL(). Он принимает аргумент URL типа CString и возвращает void. Щелкните правой кнопкой мыши в окне ClassView и выберите из контекстного меню команду **Add Function**, чтобы добавить защищенный метод TryURL(). Эта функция будет вызываться из CQueryDlg::OnQuery(), как показано в листинге 19.1. Отредактируйте OnQuery() и добавьте этот текст.

Листинг 19.1. Файл QueryDlg.cpp — CQueryDlg::OnQuery()

```
void CQueryDlg::OnQuery()
{
   const CString http = "http://";

   UpdateData(TRUE);
   m_out = "";
   UpdateData(FALSE);

   TryURL(http + m_host);
   TryURL(http + "www." + m_host);
}
```

После вызова UpdateData(TRUE) значение, введенное пользователем, помещается в m_host. Вызов UpdateData(FALSE) заполняет текстовое поле IDC_OUT, доступное только для чтения, пустым значением, присвоенным m_out. Затем следуют два вызова TryURL(). Например, если пользователь ввел microsoft.com, при первом вызове произойдет проверка адреса http://microsoft.com, а при втором — http://www.microsoft.com. Текст функции TryURL() приведен в листинге 19.2.

Листинг 19.2. Файл QueryDlg.cpp — CQueryDlg::TryURL()

```
void CQueryDlg::TryURL(CString URL)
{
   CInternetSession session;

   m_out += "Trying " + URL + "\r\n";
   UpdateData(FALSE);

   CInternetFile* file = NULL;
   try
   {
      // Мы знаем, что возвращается файл Internet,
      // поэтому преобразование типа всегда корректно.
      file = (CInternetFile*) session.OpenURL(URL);
   }
   catch (CInternetException* pEx)
   {
      // В случае неудачи присвоить file значение NULL.
      file = NULL;
      pEx->Delete();
   }
   if (file)
   {
      m_out += "Connection established. \r\n";
      CString line;

      for (int i=0; i < 20 && file->ReadString(line); i++)
      {
         m_out += line + "\r\n";
      }
      file->Close();
      delete file;
   }
   else
   {
      m_out += "No server found there. \r\n";
   }
```

```
    m_out += "------------------------\r\n";
    UpdateData(FALSE);
}
```

В оставшейся части данного раздела текст вышеприведенной функции представлен фрагментами из нескольких строк. В начале функции для запуска сеанса работы с Internet создается экземпляр класса CInternetSession. У конструктора этого класса есть аргументы, но все они имеют значения по умолчанию, которые подходят для данного приложения. Вот список этих аргументов.

- LPCTSTR pstrAgent — имя приложения. Если передается NULL, оно автоматически заполняется именем, данным при настройке AppWizard.
- DWORD dwContext — идентификатор контекста операции. При синхронной работе значение этого параметра не важно.
- DWORD dwAccessType — тип доступа — один из следующего ряда: INTERNET_OPEN_TYPE_PRECONFIG (принимаемый по умолчанию), INTERNET_OPEN_TYPE_DIRECT или INTERNET_OPEN_TYPE_PROXY.
- LPCTSTR pstrProxyName — имя proxy-сервера, если тип доступа равен INTERNET_OPEN_TYPE_PROXY.
- LPCTSTR pstr ProxyBypass — список адресов для непосредственной связи, а не через proxy-сервер, если тип доступа равен INTERNET_OPEN_TYPE_PROXY.
- DWORD dwFlags — флаги, объединяемые с помощью логического ИЛИ. Возможны значения INTERNET_FLAG_DONT_CACHE, INTERNET_FLAG_ASYNC и INTERNET_FLAG_OFFLINE.

По умолчанию значение dwAccessType берется из реестра. Очевидно, что приложение, которое рассчитано только на прямой доступ к Internet или на доступ через proxy-сервер, менее ценно, чем приложение, позволяющее пользователю задать тип доступа. Однако самостоятельная установка типа доступа к Internet может смущать пользователей. Чтобы установить доступ к Internet, используемый по умолчанию, дважды щелкните на пиктограмме Мой компьютер, затем — на Панель управления, а затем — на Internet. Выберите вкладку Connection (рис. 19.7) и заполните диалоговое окно в соответствии с вашей конфигурацией.

Если вы по причинам, обсуждаемым в главе 18, хотите установить *асинхронный* (неблокирующий) сеанс работы, значение dwFlags должно содержать INTERNET_FLAG_ASYNC. К тому же необходимо вызвать функцию-член EnableStatusCallback(), чтобы установить функцию обратного вызова. Когда при работе происходит запрос (такой как вызов OpenURL(), который передается в TryURL()), а немедленного ответа нет, неблокирующий сеанс работы возвращает код псевдоошибки ERROR_IO_PENDING. Когда ответ готов, автоматически вызывается функция обратного вызова.

В этом простом приложении нет необходимости во время ожидания ответа предоставлять пользователю возможность выполнять другую работу или взаимодействовать с интерфейсом программы, поэтому экземпляр класса CInternetSession конструируется как блокирующий сеанс, а все остальные параметры принимают свои значения по умолчанию:

```
CInternetSession session;
```

После создания объекта-сеанса работы функция TryURL() добавляет к m_out строку, в которой отображается адрес, переданный в качестве аргумента. Символы \r\n обозначают возврат каретки и перевод строки и предназначены для разделения строк в m_out. Добавленная строка выводится на экран после вызова UpdateData(FALSE):

```
m_out += "Trying " + URL + "\r\n";
UpdateData(FALSE);
```

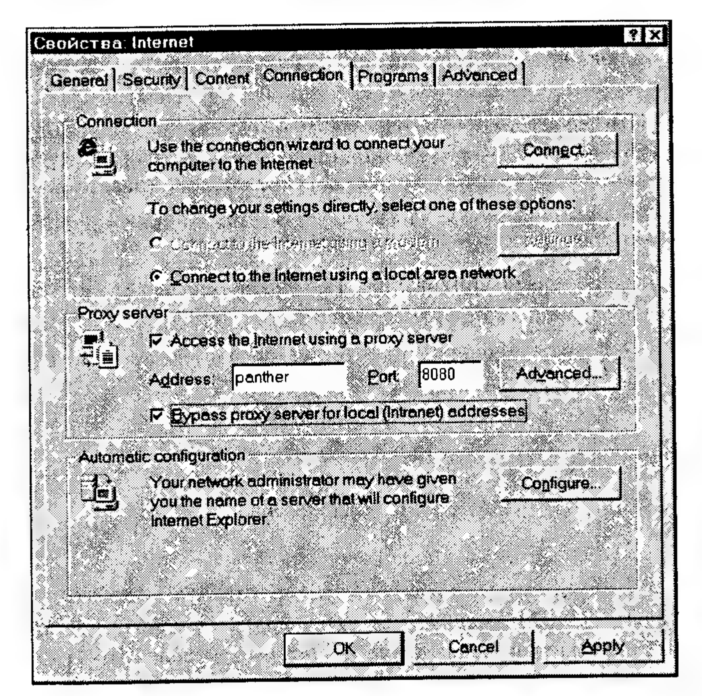


Рис. 19.7. После установки параметров соединения с Internet все приложения могут извлекать их из системного реестра

Затем следует вызов метода `OpenURL()` объекта-сеанса работы. Эта функция возвращает указатель на файл одного из четырех типов, поскольку адрес может относиться к одному из четырех следующих протоколов.

- **file://** открывает файл. Функция создает объект класса `CStdioFile` и возвращает указатель на него.
- **ftp://** осуществляет переход на сервер FTP и возвращает указатель на объект класса `CInternetFile`.
- **gopher://** осуществляет переход на сервер Gopher и возвращает указатель на объект класса `CGopherFile`.
- **http://** осуществляет переход на сервер World Wide Web и возвращает указатель на объект класса `CHttpFile`.

Поскольку и `CGopherFile`, и `CHttpFile` являются классами-наследниками `CInternetFile`, а вы уверены, что функции `TryURL()` не будет передан адрес **file://**, возвращаемый указатель можно преобразовать в тип `CInternetFile`.



В справочной документации Microsoft примеры адресов Internet показаны неправильно. Символ обратной косой черты (\) никогда не используется в адресах. Во всех примерах Microsoft вместо этого символа необходимо использовать символ прямой косой черты (/).

Если адрес невозможно открыть, переменной `file` будет присвоено значение `NULL` или `OpenURL()` сгенерирует исключение. (Исключения описаны в главе 26.) В обычном приложении невозможность открыть адрес была бы серьезной ошибкой. В этой же программе происходит всего лишь проверка работоспособности адресов, и поэтому ожидается, что некоторые из них

работать не будут. В результате это исключение необходимо перехватить просто для того, чтобы не произошла ошибка времени выполнения. В данном случае при возникновении исключения необходимо проверить, что значением переменной file является NULL. Чтобы удалить исключение и предотвратить утечку памяти, вызовите CException::Delete() — метод, который не упоминается в справке, но реально существует и безопасно удаляет исключение. Блок программы, в котором происходит вызов OpenURL(), приведен в листинге 19.3.

Листинг 19.3. Файл QueryDlg.cpp — CQueryDlg::TryURL()

```
CInternetFile* file = NULL;
try
{
   // Мы знаем, что возвращается файл Internet,
   // поэтому преобразование типа всегда корректно.
   file = (CInternetFile*) session.OpenURL(URL);
}
catch (CInternetException* pEx)
{
   // В случае неудачи присвоить file значение NULL.
   file = NULL;
   pEx->Delete();
}
```

Если file не равен NULL, эта процедура выводит фрагмент найденной страницы Web. Сначала в m_out выводится еще одна строка. Затем в цикле for с помощью вызова CInternetFile::ReadString() переменная line типа CString заполняется символами из file, пока не встретятся символы \r\n, которые отбрасываются. Считанные в line символы выводятся в m_out. Если вы хотите просматривать больше или меньше первых 20 строк на странице, измените соответствующий параметр в цикле for. После считывания первых строк файла TryURL() закрывает этот файл и удаляет его объект. Этот блок программы представлен в листинге 19.4.

Листинг 19.4. Файл QueryDlg.cpp — CQueryDlg::TryURL()

```
if (file)
{
   m_out += "Connection established. \r\n";
   CString line;

   for (int i=0; i < 20 && file->ReadString(line); i++)
   {
      m_out += line + "\r\n";
   }
   file->Close();
   delete file;
}
```

Если открыть файл не удалось, в m_out добавляется сообщение об этом:

```
else
{
   m_out += "No server found there. \r\n";
}
```

Затем, независимо от того, существовал ли файл, в m_out добавляется строка дефисов, обозначающая, что попыток доступа больше не будет, и последний вызов UpdateData(FALSE) выводит новое содержимое m_out на экран:

```
    m_out += "------------------------\r\n";
    UpdateData(FALSE);
}
```

Теперь можно откомпилировать и запустить это приложение. Если вы введете microsoft.com в текстовое поле и щелкнете на Query, то обнаружите, что страницы Web находятся и на http://microsoft.com, и на http://www.microsoft.com. Результаты выполнения этого запроса показаны на рис. 19.8.

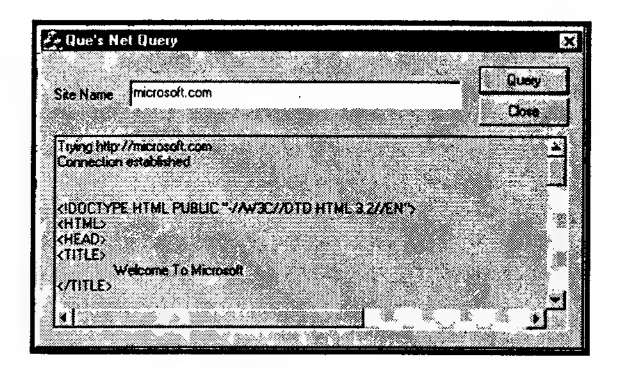


Рис. 19.8. Query находит страницы Web Microsoft

Если Query не может найти страницы Web ни по указанному имени домена, ни по этому имени с префиксом www, это еще не значит, что данный домен не существует или что у организации, которой принадлежит имя домена, нет страницы Web. Однако это уменьшает вероятность существования данной организации или ее страницы Web. Если же вы видите поток текста HTML, можете быть уверены в том, что организация существует, и у нее есть страница Web. Текст HTML можно непосредственно прочитать, а если у вас это не получается, подключитесь к данной странице с помощью броузера Web, например Microsoft Internet Explorer.

Опрос серверов FTP

В процессе изучения имени необходимо проверить, принадлежит ли оно серверу FTP. Имена большинства серверов FTP выглядят как ftp.company.com, хотя у более старых серверов форматы имен могут отличаться от этого. Проверку таких имен невозможно осуществить, просто вызвав TryURL(). Дело в том, что в TryURL() предполагается, что адрес ведет к файлу, а адреса типа ftp.greatidea.org приводят к списку файлов, который невозможно просто открыть и прочесть. Вместо дальнейшего усложнения функции TryURL() лучше создать еще один метод класса с прототипом TryFTPSite(CString host). (Для этого щелкните правой кнопкой мыши на CQueryDlg в окне ClassView и выберите Add Function. Функция будет возвращать тип void.)

Функция TryFTPSite() должна установить соедииение в рамках сеанса работы и в случае успешного соединения получить информацию, которую можно вывести в m_out, чтобы уведомить пользователя о том, что соединение произошло. Получение списка файлов является относительно сложной задачей, и, поскольку это всего лишь пример приложения, лучше выполнить более простую задачу — определить имя каталога FTP, используемого по умолчанию. Текст функции приведен в листинге 19.5.

Листинг 19.5. Файл QueryDlg.cpp — CQueryDlg::TryFTPSite()

```
void CQueryDlg::TryFTPSite(CString host)
{
   CInternetSession session;

   m_out += "Trying FTP site " + host + "\r\n";
   UpdateData(FALSE);

   CFtpConnection* connection = NULL;
   try
   {
      connection = session.GetFtpConnection(host);
   }
   catch (CInternetException* pEx)
   {
      // В случае ошибки присвоить connection значение NULL.
      connection = NULL;
      pEx->Delete();
   }
   if (connection)
   {
      m_out += "Connection established. \r\n";
      CString line;

      connection->GetCurrentDirectory(line);
      m_out += "default directory is " + line + "\r\n";

      connection->Close();
      delete connection;
   }
   else
   {
      m_out += "No server found there. \r\n";
   }

   m_out += "------------------------\r\n";
   UpdateData(FALSE);
}
```

Эта функция похожа на TryURL(), но вместо открытия файла с помощью session.OpenURL() открывается соединение FTP с помощью session.GetFtpConnection(). Исключения также перехватываются и фактически игнорируются, а обработчик исключения только застраховывает процедуру от использования указателя на соединение. Вызов функции GetCurrentDirectory() возвращает текущий каталог на удаленном сервере, в котором начинается работа. Остальная часть текста функции не изменилась по сравнению с TryURL().

Для вызова этой функции добавьте две строки в метод OnQuery():

```
TryFTPSite(m_host);
TryFTPSite("ftp." + m_host);
```

Откомпилируйте приложение и попробуйте его в работе. На рис. 19.9 показано, что Query не нашел сервер FTP microsoft.com, но нашел сервер ftp.microsoft.com. Задержка перед появлением результатов может вас раздражать. Этого можно избежать, используя асинхронное программирование гнезд или разделение программы на параллельные задачи, чтобы можно было первые результаты выводить в текстовое поле, в то время как остальные данные еще передаются по проводам. Однако в такой простой демонстрации, как эта, вам придется терпеливо ждать появления результатов. Это может занять несколько минут, в зависимости от загруженности линии, скорости передачи данных и т.д.

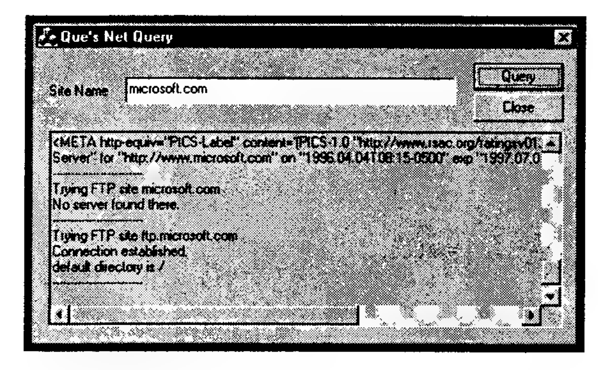


Рис. 19.9. Query нашел один сервер FTP Microsoft

Если Query не может найти страниц Web или сервера FTP, возможно, указанный домен не существует или у него нет служб Internet, кроме электронной почты. Однако есть еще несколько интересных приемов. Анализ их, несомненно, увеличит объем ваших знаний о существующих серверах.

Опрос серверов Gopher

Как и FTP, TryURL() не сработает при опросе серверов Gopher типа gopher.company.com, поскольку этот адрес возвращает не файл, а список имен файлов. Решение этой проблемы заключается в разработке функции TryGopherSite(), почти идентичной функции TryFTPSite(), за исключением того, что она открывает соединение CGopherConnection и вместо строки, описывающей каталог сервера, выводит строку, описывающую локатор Gopher, ассоциированный с сервером. Добавьте метод TryGopherSite() в класс CQueryDlg — щелкните правой кнопкой мыши в окне ClassView и выберите из контекстного меню команду Add Function, что вы уже проделали для TryFTPSite(). Текст функции TryGopherSite() приведен в листинге 19.6.

Листинг 19.6. Файл QueryDlg.cpp — CQueryDlg::TryGopherSite()

```
void CQueryDlg::TryGopherSite(CString host)
{
   CInternetSession session;

   m_out += "Trying Gopher site " + host + "\r\n";
   UpdateData(FALSE);

   CGopherConnection* connection = NULL;
   try
   {
      connection = session.GetGopherConnection(host);
   }
   catch (CInternetException* pEx)
   {
      // В случае ошибки присвоить connection значение NULL.
      connection = NULL;
      pEx->Delete();
   }

   if (connection)
   {
      m_out += "Connection established. \r\n";
```

```
        CString line;

        CGopherLocator locator = connection->CreateLocator(NULL, NULL,
GOPHER_TYPE_DIRECTORY);
        line = locator;
        m_out += "first locator is " + line + "\r\n";

        connection->Close();
        delete connection;
    }
    else
    {
        m_out += "No server found there. \r\n";
    }

    m_out += "------------------------\r\n";
    UpdateData(FALSE);
}
```

При вызове функции CreateLocator() передаются три аргумента. Первый — имя файла. В нем возможно использование символов маски имени файла. Значение NULL обозначает любой файл. Второй аргумент — селектор, который может быть равен NULL. Третий — один из следующих типов.

```
GOPHER_TYPE_TEXT_FILE
GOPHER_TYPE_DIRECTORY
GOPHER_TYPE_CSO
GOPHER_TYPE_ERROR
GOPHER_TYPE_MAC_BINHEX
GOPHER_TYPE_DOS_ARCHIVE
GOPHER_TYPE_UNIX_UUENCODED
GOPHER_TYPE_INDEX_SERVER
GOPHER_TYPE_TELNET
GOPHER_TYPE_BINARY
GOPHER_TYPE_REDUNDANT
GOPHER_TYPE_TN3270
GOPHER_TYPE_GIF
GOPHER_TYPE_IMAGE
GOPHER_TYPE_BITMAP
GOPHER_TYPE_MOVIE
GOPHER_TYPE_SOUND
GOPHER_TYPE_HTML
GOPHER_TYPE_PDF
GOPHER_TYPE_CALENDAR
GOPHER_TYPE_INLINE
GOPHER_TYPE_UNKNOWN
GOPHER_TYPE_ASK
GOPHER_TYPE_GOPHER_PLUS
```

Обычно вы не создаете локаторы для файлов или каталогов, а запрашиваете их у сервера. Локатор, возвращаемый этим вызовом CreateLocator(), описывает локатор, ассоциированный с исследуемым сервером.

Добавьте в конце метода DnQuery() пару строк, вызывающих новую функцию TryGopherSite():

```
TryGopherSite(m_host);
TryGopherSite("gopher." + m_host);
```

Сиова откомпилируйте и запустите программу. Возможно, вам придется ждать результатов несколько минут. На рис. 19.10 показано, что Query нашел два адреса Gopher для harvard.edu. В обоих случаях локатор описывает сам адрес. Этого достаточно, чтобы доказать, что по адресу harvard.edu расположен сервер Gopher, что и требуется от Query.

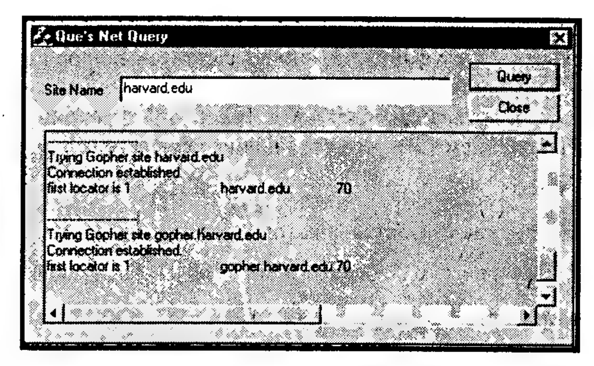


Рис. 19.10. Query нашел два адреса Gopher в Harvard



Gopher — это старый протокол, который был почти везде дополнен World Wide Web. Как правило, если сервер поддерживает протокол Web, он существовал в Internet до появления World Wide Web (1989), или, по крайней мере, до начала ее быстрого распространения (1992). Более того, этот сервер, скорее всего, был достаточно велик в начале 1990-х, чтобы иметь администратора, который настроил меню и текст Gopher.

Отправка запроса Finger с помощью Gopher

Информацию об адресе может дать еще один протокол. Это одии из самых старых протоколов Internet, и он называется *Finger*. С его помощью можно получить информацию об отдельном пользователе или целом сервере, и, хотя на многих серверах этот протокол отключен, еще больше серверов в ответ на запрос Finger предоставит полезную информацию.

Класса MFC или функции API, в названия которых входило бы слово *finger*, не существует, но это не значит, что невозможно применять уже описанные классы. В данном разделе используется один фокус, основанный на знании протоколов Finger и Gopher. Хотя классы Winlnet рассчитаны, в основном, на начинающих программистов Internet, они также могут предложить многое и опытным программистам, которые знают, что происходит за кулисами.

Как описано в главе 18, все операции Internet используют и компьютер, и порт. Популярные службы используют стандартные номера портов. Например, если методу `CInternetSession::OpenURL()`передан адрес, начинающийся с **http://**, то в нем запрограммирована связь с портом 80 на удаленном компьютере. При вызове `GetFtpConnection()` связь происходит с портом 21. Gopher использует порт 70. На рис. 19.10 видно, что в локаторе, описывающем `gopher.harvard.edu`, упоминается порт 70.

В документации при описании функционирования Gopher сказано следующее: если вы создадите локатор с именем компьютера, портом 70, типом Gopher 0 (константа `GOPHER_TYPE_TEXT_FILE` равна 0) и строкой имени файла, любой клиент Gopher просто отошлет эту строку в порт 70, независимо от того, является ли она именем файла. Сервер Gopher, принимающий данные по этому порту, в ответ посылает указанный файл.

Finger — также простой протокол. На отправку строки в порт 79 удаленного компьютера сервер Finger отреагирует отправкой ответа. Если строка состоит только из символов `\r\n`, ответом обычно является список всех пользователей на этом компьютере и какая-то другая информация о них, например настоящие имена. (На многих серверах считают, что это является нарушением личных прав или безопасности, и протокол Finger на них отключен. Однако на других серверах эта информация намеренно помещается на страницы Web.)

Подводя итоги, можем отметить, что, если построить локатор Gopher с портом 79 вместо стандартного 70 и пустым именем файла, можно осуществить запрос Finger с помощью классов WinInet. Для начала добавьте в класс CQueryDlg функцию TryFinger(), которая имеет аргумент host типа CString и возвращает тип void. Текст этой функции очень похож на текст TryGopherSite(), за исключением того, что соединение осуществляется через порт 79:

```
connection = session.GetGopherConnection(host,NULL,NULL,79);
```

После установки соединения создается локатор текстового файла:

```
CGopherLocator locator = connection->CreateLocator(NULL, NULL, GOPHER_TYPE_TEXT_FILE);
```

На этот раз вместо простого преобразования в тип CString он используется для открытия файла:

```
CGopherFile* file = connection->OpenFile(locator);
```

Затем на экран выводятся первые 20 строк этого файла, точно так же, как в TryURL() выводились первые 20 строк файла, возвращаемого сервером Web. Соответствующий текст программы приведен в листинге 19.7.

Листинг 19.7. Файл QueryDlg.cpp — CQueryDlg::TryFinger() (фрагмент)

```
if (file)
{
   CString line;

   for (int i=0; i < 20 && file->ReadString(line); i++)
   {
      m_out += line + "\r\n";
   }
   file->Close();
   delete file;
}
```

Целиком функция TryFinger() показана в листинге 19.8.

Листинг 19.8. Файл QueryDlg.cpp — CQueryDlg::TryFinger()

```
void CQueryDlg::TryFinger(CString host)
{
   CInternetSession session;

   m_out += "Trying to Finger " + host + "\r\n";
   UpdateData(FALSE);

   CGopherConnection* connection = NULL;

   try
   {
      connection = session.GetGopherConnection(host,NULL,NULL,79);
   }
   catch (CInternetException* pEx)
   {
      // В случае ошибки присвоить connection значение NULL.
      connection = NULL;
      pEx->Delete();
   }
   if (connection)
   {
      m_out += "Connection established. \r\n";
```

```
    CGopherLocator locator = connection->CreateLocator(NULL, NULL,
GOPHER_TYPE_TEXT_FILE);
    CGopherFile* file connection->OpenFile(locator);
    if (file)
    {
       CString line;

       for (int i=0; i < 20 && file->ReadString(line); i++)
       {
          m_out += line + "\r\n";
       }
       file->Close();
       delete file;
    }

    connection->Close();
    delete connection;
  }
  else
  {
    m_out += "No server found there. \r\n";
  }

  m_out += "------------------------\r\n";
  UpdateData(FALSE);

}
```

Добавьте в конце метода OnQuery() строку, вызывающую новую функцию:

```
TryFinger(m_host);
```

Теперь откомпилируйте и запустите приложение. На рис. 19.11 показан результат опроса домена whitehouse.gov, в конце которого находится раздел Finger.

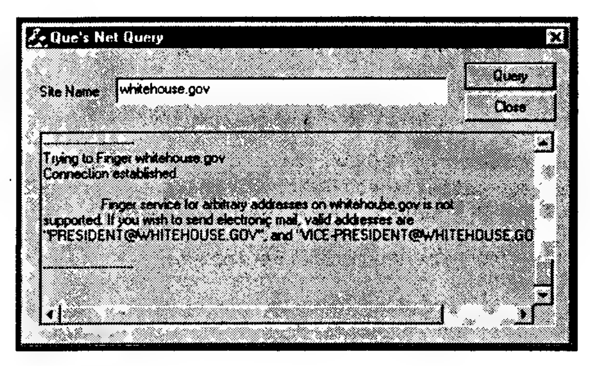


Рис. 19.11. Query получает адреса электронной почты от сервера Finger Белого дома

На заметку Если на исследуемом домене не работает сервер Finger, задержка будет дольше обычной и появится диалоговое окно, сообщающее об истечении времени попытки соединения. Щелкните на кнопке OK этого окна, если оно появится.

Отправка запроса Whois с помощью Gopher

Информацию об адресах предоставляет еще один — последний — протокол. Он также является старым протоколом, не поддерживаемым напрямую классами WinInet. Он называется *Whois*, и такая служба предоставляется всего несколькими серверами во всей Internet. Серверы, предлагающие эту службу, обслуживаются организациями, которые регистрируют имена доменов. Например, имена доменов, заканчивающиеся на `.com`, регистрирует организация, которая называется InterNIC, и ей принадлежит сервер Whois `rs.internic.net` (*rs* обозначает *Registration Services — службы регистрации*). Подобно Finger, Whois реагирует на строку, посланную в его порт; номер порта Whois — 43. В отличие от Finger в локаторе посылается не пустая строка, а имя компьютера, о котором необходимо получить информацию. Подключение происходит всегда к `rs.internic.net`. (Выделенные серверы Whois позволяют избежать этого, но на практике никто так не поступает.)

Добавьте функцию с именем `TryWhois()`; как обычно, она принимает аргумент `host` типа `CString` и возвращает тип `void`. Текст функции приведен в листинге 19.9.

Листинг 19.9. Файл `QueryDlg.cpp` — `CQueryDlg::TryWhois()`

```
void CQueryDlg::TryWhois(CString host)
{
   CInternetSession session;

   m_out += "Trying Whois for " + host + "\r\n";
   UpdateData(FALSE);

   CGopherConnection* connection = NULL;
   try
   {
      connection = session.GetGopherConnection("rs.internic.net",NULL,NULL,43);
   }
   catch (CInternetException* pEx)
   {
      // В случае ошибки присвоить connection значение NULL.
      connection = NULL;
      pEx->Delete();
   }
   if (connection)
   {
      m_out += "Connection established. \r\n";
      CGopherLocator locator = connection->CreateLocator(NULL, host,
GOPHER_TYPE_TEXT_FILE);
      CGopherFile* file = connection->OpenFile(locator);
      if (file)
      {
         CString line;
         for (int i=0; i < 20 && file->ReadString(line); i++)
         {
            m_out += line + "\r\n";
         }
         file->Close();
         delete file;
      }
      connection->Close();
      delete connection;
   }
   else
   {
      m_out += "No server found there. \r\n";
```

```
    }
    m_out += "------------------------\r\n";
    UpdateData(FALSE);
}
```

Добавьте в конце OnQuery() строку, вызывающую эту функцию:

```
TryWhois(m_host);
```

Откомпилируйте и запустите приложение последний раз. На рис. 19.12 показана часть Whois-отчета о mcp.com — домене издательства Macmillan Computer Publishing, родительской компании Que.

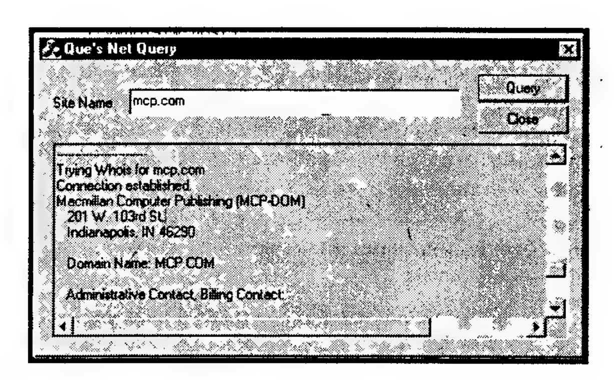


Рис. 19.12. Query получает реальные адреса и имена с сервера Whois InterNIC

Теперь, когда добавлен фрагмент программы, отвечающий за работу с Finger, нельзя больше игнорировать невозможность связаться по протоколу Finger. Если при вызове OpenFile() в TryFinger() происходит попытка открыть файл на компьютере, на котором не работает сервер Whois, то возникает следующая проблема: управление не возвращается OnQuery() и TryWhois() никогда не будет вызвана. Чтобы предотвратить эту ситуацию, необходимо заключить вызов OpenFile() в блок try/catch. В листинге 19.10 показаны необходимые изменения.

Листинг 19.10. Файл QueryDlg.cpp — изменения в TryFinger()

```
// Строку
      CGopherFile* file connection->OpenFile(locator);
// необходимо заменить следующими:
      CGopherFile* file = NULL;
      try
      {
         file = connection->OpenFile(locator);
      }
      catch (CInternetException* pEx)
      {
         // В случае ошибки присвоить file значение NULL.
         file = NULL;
         pEx->Delete();
      }
```

Отредактируйте TryFinger(), снова откомпилируйте Query и опросите адрес, по которому нет сервера Finger, например microsoft.com. Программа должна успешно дойти до части, работающей с Whois.

Над чем еще стоит поработать

Приложение Query, написанное в данной главе, выполняет многое, но оно могло бы делать еще больше. Существуют протоколы электронной почты и новостей, доступ к которым можно получить, немного расширив классы WinInet и использовав их для соединения со стандартными портами этих служб. Также можно связаться с некоторыми известными поисковыми системами Web и выполнить запрос, сформировав адреса в соответствии с шаблонами, используемыми в этих системах. Таким образом можно автоматизировать "заглядывание" в Internet, которым большинство из нас занимается, когда нас интересует определенное имя домена или организация. Если вы хотите узнать больше о протоколах Internet, номерах портов и о том, что происходит, когда клиент соединяется с сервером, рекомендуем прочесть книгу *Building Internet Applications with Visual C++*, выпущенную Que. Она была написана для Visual C++ 2.0, и хотя все приложения из книги компилируются и запускаются с последними версиями MFC, сейчас их можно было бы написать значительно быстрее. Однако основная ценность книги состоит в освещении работы протоколов.

Возможности классов WinInet также намного разнообразнее, чем описано в этой главе. Приложение Query не использует их для получения реальных файлов по Internet. В комплект Visual C++ 5.0 включены два демонстрационных приложения WinInet, которые используются для получения файлов:

- FTPTREE (строит древовидный список файлов и каталогов сервера FTP);
- TEAR (получает с сервера Web страницу HTML).

Microsoft анонсирует выход новых программных продуктов на протяжении следующих нескольких месяцев. Следите за появлением на сервере Web **www.microsoft.com** библиотек и других материалов разработчиков программного обеспечения, еще более облегчающих и ускоряющих программирование для Internet.

ГЛАВА 20

Создание элемента управления ActiveX для Internet

В этой главе...

Внедрение элементов управления ActiveX в страницы Web Microsoft Explorer

Внедрение элемента управления ActiveX в страницы Web Netscape Navigator

Регистрация безопасности элемента управления

Выбор между ActiveX и Java

Повышение быстродействия элементов управления ActiveX

Ускорение работы элементов управления с помощью асинхронных свойств

В главе 17 речь шла о создании элементов управления и включении их в приложения, написанные на Visual Basic, Visual C++ и макроязыке VBA. Есть еще одна возможность использования самостоятельно разработанных элементов управления — включение их в страницы Web. Но элементы управления ActiveX, сгенерированные старыми версиями Visual C++, слишком громоздки (в смысле кода программы) и "медленны", чтобы помещать их на страницы Web. В этой главе показано, как написать их более быстродействующие и компактные версии и как вставить такие элементы управления в страницы Web.

Внедрение элементов управления ActiveX в страницы Web Microsoft Explorer

Вставить элемент управления ActiveX в страницу Web очень легко, если есть уверенность, что эта страница будет загружаться в Microsoft Explorer версии 3.0 или более поздней. Для этого используется дескриптор <OBJECT> — относительно недавнее дополнение языка HTML, с помощью которого можно описывать большой спектр объектов, вставляемых в страницы Web: видеоклипы, звук, программы Java, элементы управления ActiveX и многие другие виды информации и средства взаимодействия с пользователем. В листинге 20.1 показан исходный текст HTML-страницы, в которой выводится элемент управления Dieroll из главы 17.

Листинг 20.1. Файл fatdie.html — использование <OBJECT>

```
<HEAD>
<TITLE>Страница Web с игральной костью</TITLE>
</HEAD>
<BODY>
<OBJECT ID="Dieroll1"
CLASSID="CLSID:46646B43-EA16-11CF-870C-00201801DDD6"
CODEBASE="dieroll.cab#Version=1,0,0,1"
WIDTH="200"
HEIGHT="200">
<PARAM NAME ="ForeColor" VALUE="0">
<PARAM NAME ="BackColor" VALUE="16777215">
Если вы видите этот текст, ваш броузер не поддерживает дескриптор OBJECT.
<BR>
</OBJECT>
<BR>
А это текст после игральной кости
</BODY>
</HTML>
```

Здесь единственная неприятная вещь — CLSID, и поскольку вы являетесь разработчиком программы, можете его получить — вырезать из файла dieroll.odl (библиотеки описания объекта — Object Description Library). Для того чтобы быстро открыть dieroll.odl, воспользуйтесь FileView. Вот фрагмент dieroll.odl, в котором содержится CLSID:

```
// Class information for CDierollCtrl

[ uuid(46646B43-EA16-11CF-870C-00201801DDD6),
  helpstring("Dieroll Control"), control ]
```

Этот фрагмент находится в конце файла dieroll.odl — предыдущие CLSID относятся не ко всему элементу управления, а к его частям. Скопируйте uuid, заключенный в круглые скобки, в текст HTML.



Microsoft выпускает продукт под названием Control Pad, с помощью которого можно получить CLSID из реестра. Он облегчает жизнь тем создателям страниц Web, кого пугают указания наподобие "откройте файл ODL" или у которых нет файла ODL, поскольку он не поставляется с элементом управления. Однако так как именно вы создаете данный элемент управления и знаете, как открывать файлы в Visual Studio, в настоящей главе Control Pad не описывается. Если вас заинтересовал этот продукт, обратитесь на страницу Web, посвященную Microsoft Control Pad по адресу **http://www.microsoft.com/workshop/author/cpad/**.

Атрибут CODEBASE дескриптора OBJECT указывает, где находится файл OCX, чтобы в том случае, если у пользователя нет экземпляра элемента управления ActiveX, последний был загружен автоматически. Использование CLSID означает, что, если пользователь уже установил этот элемент управления ActiveX, время на загрузку не тратится, а элемент управления сразу используется. Можно просто указать адрес файла OCX, но, чтобы автоматизировать загрузку библиотек, атрибут CODEBASE указывает на файл CAB. Размещение элемента управления в файле CAB уменьшит время загрузки почти в два раза. Больше о технологии CAB можно узнать по адресу **http://www.microsoft.com/intdev/cab/**. Эта страница написана для разработчиков, использующих Java, но данная технология весьма полезна и для уменьшения времени загрузки элементов управления ActiveX.



Если у вас нет доступа к серверу Web, на котором можно разместить элементы управления во время их разработки, используйте в атрибуте CODEBASE адрес вида file://, указывающий на местоположение элемента управления на жестком диске.

Остальные атрибуты дескриптора OBJECT должны быть интуитивно понятны, если вы прежде создавали страницы Web: ID используется в других дескрипторах этой страницы для ссылки на данный элемент управления, WIDTH и HEIGHT указывают размер элемента управления в пикселях, а HSPACE и VSPACE — горизонтальный и вертикальный отступы вокруг элемента управления, указываемые также в пикселях.

Все, что заключено между дескрипторами <OBJECT...> и </OBJECT>, игнорируется броузерами, распознающими дескриптор OBJECT. (Дескриптор <OBJECT...> обычно занимает много строк и содержит всю описывающую объект информацию.) Броузеры, не распознающие дескриптор OBJECT, игнорируют <OBJECT...> и </OBJECT> и выводят текст HTML между ними (в данном случае — строку с сообщением, что броузер не поддерживает дескриптор). Это — часть спецификации броузера Web: броузер должен игнорировать дескрипторы, которые он не может распознать, т.е. воспринимать их, как обычный текст.

На рис. 20.1 показана эта страница, загруженная в Microsoft Explorer 3.0. После щелчка мышью игральная кость начинает вращаться. Это выглядит просто и впечатляюще, но немедленно обнаруживаются два недостатка этого метода.

- Не все броузеры поддерживают дескриптор OBJECT.
- Загрузка элемента управления может занять много времени.

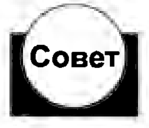
На рис. 20.2 показана эта же страница Web, загруженная в Netscape Navigator 2.0. Он не поддерживает дескриптор OBJECT и поэтому не отображает игральную кость. А Netscape Navigator использует больше половины пользователей, работающих с Web! Значит ли это, что писать элементы управления ActiveX для страниц Web не стоит? Вовсе нет. Как вы узнаете из следующего раздела, есть способ предоставить и пользователям Navigator, и пользователям Explorer возможность применять одни и те же элементы управления.

Существенной проблемой является размер выполняемого кода элемента управления. Окончательная версия элемента управления Dieroll, созданного в главе 17, занимает 26 Кбайт. Многие разработчики ограничивают объем графики и прочего загружаемого материала значением 50 Кбайт, а этот простой элемент управления занимает половину данного

объема. Более мощные элементы управления с легкостью превысят предел. Большая часть этой главы посвящена минимизации размера выполняемого кода элементов управления ActiveX и другим способам уменьшения времени их загрузки. Ваши усилия окупятся сторицей, поскольку разработчики страниц Web смогут использовать созданные вами элементы управления в полную силу, не заботясь о том, что пользователи назовут их страницы "медленными" (а это самое худшее, что можно сказать о любой странице Web).

Рис. 20.1. Microsoft Internet Explorer показывает элементы управления ActiveX

Есть и третий недостаток, которого вы не заметите, поскольку на вашем компьютере установлен Visual C++. Этот элемент управления требует наличия библиотеки MFC. Для того чтобы он работал, пользователь должен загрузить и установить эту библиотеку. Механизм, автоматически загружающий и устанавливающий элементы управления, не делает этого с библиотекой, хотя это и возможно с использованием файла CAB, о котором шла речь выше.



Пример страницы Web, включающей файл CAB для элемента управления Dieroll и библиотеки MFC, можно найти на узле **http://www.gregcons.com/dieroll.htm**.

На заметку

Возможно, вам придет в голову идея скомпоновать библиотеку MFC и элемент управления статически. На первый взгляд, сделать это несложно — выбрать Build⇒Settings. На вкладке General есть раскрывающийся список, позволяющий задать статическую компоновку. Сделав это, вы столкнетесь с множеством ошибок компоновки: функций классов COleControl и CPropPage. Последние не входят в состав DLL-модулей, которые можно компоновать статически. (Это происходит потому, что в Microsoft не рассчитывали, что кому-то может прийти в голову идея статически ком-

поновать библиотеки MFC в элементы управления.) Установка другой библиотеки, с помощью которой можно было бы статически прикомпоновать эти функции, выходит за пределы этой главы, тем более что это повлечет за собой создание огромного (более 1 Мбайт) выполняемого файла элемента управления, загрузка которого заняла бы слишком много времени.

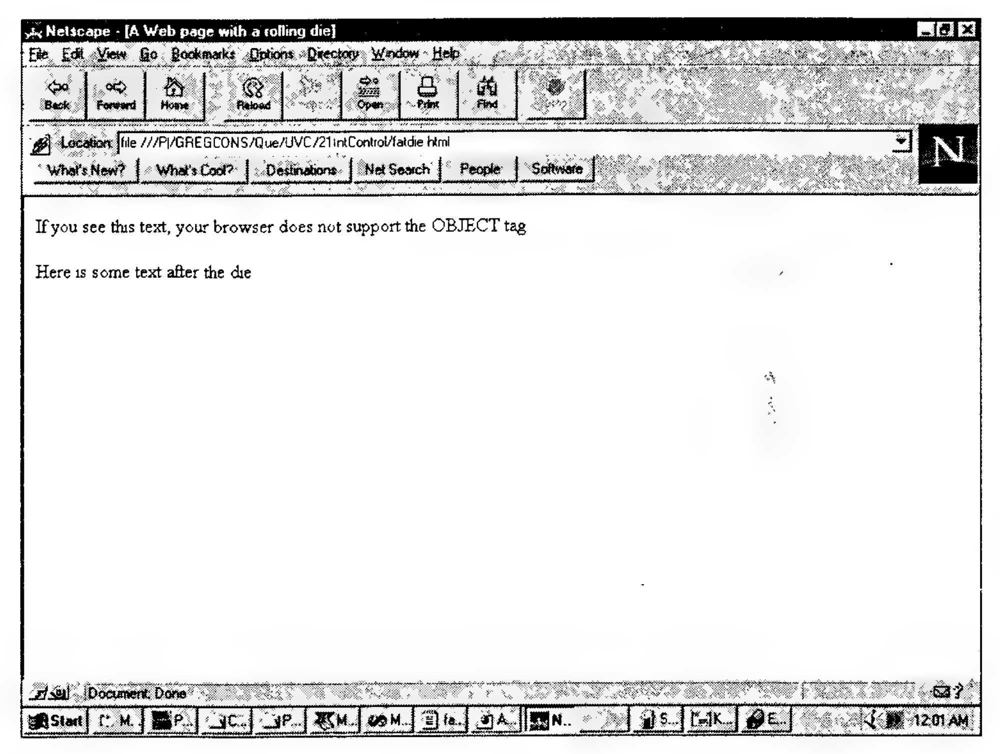


Рис. 20.2. Netscape Navigator не показывает элементы управления ActiveX

Внедрение элемента управления ActiveX в страницы Web Netscape Navigator

Компания *NCompass Labs* (**www.ncompasslabs.com**) разработала надстройку для Netscape под названием `ControlActive`, позволяющую внедрять элементы управления ActiveX в страницы, предназначенные для просмотра с помощью Netscape Navigator. Модифицированный HTML-текст страницы приведен в листинге 20.2. (Удержитесь от искушения загрузить этот файл в Netscape до тех пор, пока вы не зарегистрируете элемент управления как безопасный для инициализации и обработки в сценарии. Как это сделать, будет описано в следующем разделе.)

Листинг 20.2. Файл fatdie2.html — использование <OBJECT> и <EMBED>

```
<HTML>
<HEAD>
<TITLE>Страница Web с игральной костью</TITLE>
</HEAD>
<BODY>
<OBJECT ID="Dieroll1"
CLASSID="CLSID:46646B43-EA16-11CF-870C-00201801DDD6"
CODEBASE="dieroll.cab#Version=1,0,0,1"
WIDTH="200"
HEIGHT="200">
<PARAM NAME="ForeColor" VALUE="0">
<PARAM NAME="BackColor" VALUE="16777215">
<PARAM NAME="Image" VALUE="beans.bmp">
<EMBED LIVECONNECT NAME="Dieroll1"
WIDTH="200"
HEIGHT="200"
CLASSID="CLSID:46646B43-EA16-11CF-870C-00201801DDD6"
CODEBASE="dieroll.cab#Version=1,0,0,1"
PARAM_ForeColor="0"
PARAM_BackColor="16777215">
</OBJECT>
<BR>
А это текст после игральной кости
</BODY>
</HTML>
```

Надстройка активизируется дескриптором <EMBED>. Поскольку этот дескриптор находится между дескрипторами <OBJECT> ... </OBJECT>, Microsoft Internet Explorer и другие броузеры, распознающие дескриптор OBJECT, игнорируют дескриптор EMBED. Это означает, что исходный текст HTML будет одинаково выводить элемент управления и в Netscape Navigator, и в Explorer. Возможно, вам захочется включить в свою страницу ссылку на страницу NCompass, чтобы помочь читателям приобрести надстройку и узнать о ней больше.

Microsoft собирается адаптировать элементы управления ActiveX для всех платформ и броузеров, которые, в соответствии с их лозунгом, "оживят Internet"[3]. Спецификация элемента ActiveX более не является частным документом. Она передана комитету, который будет следить за соответствием стандарту. Поэтому не обращайте внимания на тех горе-специалистов, которые утверждают, что элементы управления ActiveX необходимо создавать только в том случае, если читатели ваших страниц пользуются Internet Explorer!

Регистрация безопасности элемента управления

Для всех пользователей, работающих при среднем уровне безопасности (Medium Safety level), элемент управления должен быть зарегистрирован как безопасный. Для тех, кто будет просматривать страницу с таким элементом управления, это служит гарантией, что независимо от того, какие функции вызываются в сценарии и какие параметры инициализируются с помощью атрибута PARAM, ничего опасного не произойдет. За примером опасного элемента управления далеко ходить не надо — им может послужить элемент управления, удаляющий

[3] В английском оригинале есть плохо переводимая на русский язык игра слов. Лозунг Microsoft гласит — *Activate Internet*, а средством для этого должна служить технология *ActiveX*. — *Прим. ред.*

при запуске файлы на вашем компьютере. Страница, поместившая этот элемент управления в сценарии или инициализировавшая имя параметра с помощью атрибута PARAM, может заставить элемент управления удалить очень важный файл или файлы, основываясь на предположении о том, где пользователи обычно хранят документы. Например, несложно удалить файл C:\MSOFFICE\WINWORD\WINWORD.EXE, но у пользователей Word из-за этого возникнут кое-какие проблемы. Когда при среднем уровне безопасности загружается страница с элементом управления, не зарегистрированным как безопасный, Internet Explorer выводит сообщение об ошибке. Надстройка ControlActive фирмы NCompass Labs также отказывается загружать элементы управления, не зарегистрированные как безопасные.

Для того чтобы выполнить регистрацию, необходимо сначала добавить три функции в DierollCtl.cpp. (Они взяты без изменения из SDK ActiveX.) Вызов таких функций происходит в тексте программы, представленном ниже в данном разделе. Не забудьте добавить описания этих функций в файл заголовка. Текст функций приведен в листинге 20.3.

Листинг 20.3. Файл DierollCtl.cpp — новые функции, отмечающие элемент управления как безопасный

```
///////////////////////////////////////////////////////////////////
// Скопировано из SDK ActiveX.
// С помощью этих функций элемент управления
// регистрируется как безопасный.

HRESULT CreateComponentCategory(CATID catid, WCHAR* catDescription)
{

    ICatRegister* pcr = NULL ;
    HRESULT hr = S_OK ;

    hr = CoCreateInstance(CLSID_StdComponentCategoriesMgr,
       NULL, CLSCTX_INPROC_SERVER, IID_ICatRegister,
      (void**)&pcr);
       if (FAILED(hr))
           return hr;

    // Убедиться в том, что ключ
    // HKCR\Component Categories\{..catid...} зарегистрирован.
    CATEGORYINFO catinfo;
    catinfo.catid = catid;
    catinfo.lcid = 0x0409 ; // english

       // Проверить, не является ли слишком длинным переданное описание.
       // Если оно слишком длинное, скопировать только первые 127 символов.
       int len = wcslen(catDescription);
       if (len>127)
           len = 127;
    wcsncpy(catinfo.szDescription, catDescription, len);
       // Описание должно завершаться нулевым символом.
       catinfo.szDescription[len] = \0;

    hr = pcr->RegisterCategories(1, &catinfo);
       pcr->Release();

       return hr;
}


HRESULT RegisterCLSIDInCategory(REFCLSID clsid, CATID catid)
{
```

```
    // Зарегистрировать информацию о категориях компонентов.
  ICatRegister* pcr = NULL ;
  HRESULT hr = S_OK ;
  hr = CoCreateInstance(CLSID_StdComponentCategoriesMgr,
     NULL, CLSCTX_INPROC_SERVER, IID_ICatRegister,
    (void**)&pcr);
  if (SUCCEEDED(hr))
  {
     // Зарегистрировать категорию как "реализованную" классом.
     CATID rgcatid[1] ;
     rgcatid[0] = catid;
     hr = pcr->RegisterClassImplCategories(clsid, 1, rgcatid);
  }

  if (pcr != NULL)
     pcr->Release();

     return hr;
}

HRESULT UnRegisterCLSIDInCategory(REFCLSID clsid, CATID catid)
{
   ICatRegister* pcr = NULL ;
   HRESULT hr = S_OK ;
   hr = CoCreateInstance(CLSID_StdComponentCategoriesMgr,
      NULL, CLSCTX_INPROC_SERVER, IID_ICatRegister, (void**)&pcr);
   if (SUCCEEDED(hr))
   {
     // Отменить регистрацию категории как "реализованной"
     // данным классом.
     CATID rgcatid[1] ;
     rgcatid[0] = catid;
     hr = pcr->UnRegisterClassImplCategories(clsid, 1, rgcatid);
   }

   if (pcr != NULL)
      pcr->Release();

      return hr;
}
```

Затем добавьте в начало файла `DierollCtl.cpp` две директивы `#include`:

```
#include "comcat.h"
#include "objsafe.h"
```

Наконец, измените функцию `UpdateRegistry()` в файле `DierollCtl.cpp`, чтобы в ней происходили вызовы новых функций. Измененный текст программы вызывает функцию `CreateComponentCategory()`, чтобы создать категорию `CATID_SafeForScripting`, и добавляет элемент управления в эту категорию. Затем он создает категорию `CATID_SafeForInitializing` и также добавляет в нее элемент управления. В листинге 20.4 показана новая версия `UpdateRegistry()`.

Листинг 20.4. Файл `DierollCtl.cpp` — `CDierollCtrl::CDierollCtrlFactory::UpdateRegistry()`

```
BOOL CDierollCtrl::CDierollCtrlFactory::UpdateRegistry(BOOL bRegister)
{
   // TODO: проверьте, чтобы элемент управления соответствовал
   // правилам apartment-threading model. Обратитесь к MFC TechNote 64,
   // если вам необходима дополнительная информация. Если элемент управления
```

```
    // не соответствует этим правилам, необходимо изменить нижеследующий
    // код, заменив шестой параметр - вместо
    // afxRegInsertable | afxRegApartmentThreading
    // использовать afxRegInsertable.

    if (bRegister)
    {
      HRESULT hr = S_OK ;

      // Зарегистрировать как безопасный для обработки сценария.
      hr = CreateComponentCategory(CATID_SafeForScripting,
            L"Controls that are safely scriptable");

      if (FAILED(hr))
         return FALSE;

      hr = RegisterCLSIDInCategory(m_clsid, CATID_SafeForScripting);

      if (FAILED(hr))
         return FALSE;

      // Зарегистрировать как безопасный для инициализации.
      hr = CreateComponentCategory(CATID_SafeForInitializing,
            L"Controls safely initializable from persistent data");

      if (FAILED(hr))
         return FALSE;

      hr = RegisterCLSIDInCategory(m_clsid, CATID_SafeForInitializing);

      if (FAILED(hr))
         return FALSE;

      return AfxOleRegisterControlClass(
         AfxGetInstanceHandle(),
         m_clsid,
         m_lpszProgID,
         IDS_DIEROLL,
         IDB_DIEROLL,
         afxRegInsertable | afxRegApartmentThreading,
         _dwDierollOleMisc,
         _tlid,
         _wVerMajor,
         _wVerMinor);
    }
      else
      {
         HRESULT hr = S_OK ;

         hr = UnRegisterCLSIDInCategory(m_clsid, CATID_SafeForScripting);

         if (FAILED(hr))
            return FALSE;

         hr = UnRegisterCLSIDInCategory(m_clsid, CATID_SafeForInitializing);

         if (FAILED(hr))
            return FALSE;

         return AfxOleUnregisterClass(m_clsid, m_lpszProgID);
      }
}
```

Чтобы убедиться в том, что это работает, откройте Explorer и установите средний уровень безопасности, а затем загрузите страницу HTML с прежней версией элемента управления.

Должно появиться предупреждение о небезопасности элемента управления. Теперь внесите указанные выше изменения, откомпилируйте элемент управления и снова загрузите страницу. На этот раз предупреждение не появится.

Выбор между ActiveX и Java

Элементы управления ActiveX можно при желании разрабатывать и на языке приложений Java, например, с помощью Microsoft Visual J++, интегрированного в Visual Studio. Но когда программисты говорят о преимуществах ActiveX или Java, они обычно сравнивают ActiveX и специальные аплеты (applets) Java для Web — небольшие компактные приложения, которые запускаются на страницах Web, но не работают самостоятельно.

Многих пользователей беспокоит безопасность запуска приложений, написанных не ими, если они не знают, какая организация или разработчик поставляют это приложение. В языке Java используется подход, ограничивающий действия, которые могут выполнять приложения. Поэтому даже "злонамеренные" приложения не могут причинить реального вреда. Однако регулярно появляющиеся сообщения о недостатках ограничительного подхода подрывают доверие к Java. Но даже если безопасность приложений Java и будет гарантирована, те же ограничения не позволяют приложениям выполнять некоторые полезные задачи.

Подход, выбранный Microsoft для технологии ActiveX, основан на доверии к разработчику. Его можно распространить также на Java-программы и на любые другие программы такого рода. Каждая программа включает электронную подпись, так что пользователь всегда располагает сведениями о разработчике и может быть уверен, что с момента фиксации подписи данная программа не была изменена. Конечно, это не исключает возможности возникновения неприятностей при запуске программы, но, по крайней мере, вы всегда знаете, кого винить в таком случае. Это похоже на покупку программных продуктов в компьютерном магазине. Более подробная информация по этому вопросу доступна на странице **http://www.microsoft.com/ie/most/howto/trusted.htm**, где можно найти многочисленные ссылки.

Наверное, самое большое различие между технологией ActiveX и аплетами Java заключается в их загрузке. Код программы Java загружается каждый раз при загрузке страницы, содержащей ее. Код элемента управления ActiveX загружается только один раз, если он не был установлен другим способом (например, с компакт-диска). Копия этого элемента управления хранится на компьютере пользователя и заносится в реестр Registry. Размер загружаемого кода Java невелик, поскольку его большая часть содержится в *Java Virtual Machine* (виртуальной машине Java), установленной на компьютере, скорее всего, как часть броузера.

Загружаемый код ActiveX может быть намного больше, хотя с помощью оптимизации, рассмотренной в следующей главе, можно значительно уменьшить его размер, опираясь на библиотеки и прочие программные компоненты, уже присутствующие в компьютере. Если пользователи загружают страницу только один раз, их может раздражать то, что элементы ActiveX занимают дисковое пространство и место в реестре. С другой стороны, если некоторый пользователь часто посещает определенную страницу, его вполне устроит тот факт, что не тратится время на повторную загрузку. Элемент управления просто активизируется и работает.

Есть и другие различия. Аплеты Java не могут генерировать сообщения контейнеру о том, что что-то произошло. Аплеты Java невозможно лицензировать, и в них часто не делается различие между разработкой и использованием в готовом виде. Аплеты Java в отличие от элементов управления ActiveX невозможно использовать в формах Visual Basic, программах Visual C++ и документах Word. Элементы управления ActiveX работают примерно в 10 раз быстрее аплетов Java. В пользу аплетов Java можно высказать следующие соображения — они являются переносимыми с одной платформы на другую, а размер их обычно меньше, чем размер эквивалентных по функциям элементов управления ActiveX.

Повышение быстродействия элементов управления ActiveX

Microsoft не предполагала, что элементы управления OCX будут размещаться на страницах Web, и от того, что их название изменилось на ActiveX, их объем не уменьшился и скорость работы не повысилась. Поэтому AppWizard из пакета Visual C++ предлагает несколько параметров для оптимизации элементов управления. В данной главе в качестве примера изменяются параметры уже созданного элемента управления Dieroll. Поскольку размер Dieroll и так невелик и он загружается быстро, простые изменения не дадут большого эффекта. Однако стоит изучить эти приемы, чтобы в дальнейшем применять их при разработке других элементов управления, объем которых наверняка будет больше, чем объем Dieroll.

Несколько первых параметров, влияющих на размер элемента управления, доступны на этапе 2 настройки ActiveX ControlWizard.

- Активизируется при появлении (флажок Activate when visible)
- Невидим во время выполнения (флажок Invisible at runtime)
- Доступен в диалоговом окне Insert Option (флажок Available in Insert Option dialog)
- Имеет диалоговое окно About (флажок Has an "About" box)
- Действует как простой элемент управления фреймом (флажок Acts as a simple frame control)

Если вы разрабатываете элемент управления только для Web, часть из этих параметров можно игнорировать. Например, не имеет значения, есть ли у элемента управления окно About; пользователи все равно не смогут его открыть во время просмотра страницы Web.

Параметр активизации при появлении очень важен. Активизация элемента управления связана с довольно большим объемом работы, и ее необходимо по возможности отложить на потом, чтобы элемент управления загружался быстрее. Если он активизируется сразу при появлении, время загрузки увеличится. Чтобы изменить такую методику работы существующей программы Dieroll, откройте в Visual Studio проект Dieroll, а затем откройте с помощью FileView файл DierollCtl.cpp. Отыщите фрагмент текста, приведенный в листинге 20.5.

Листинг 20.5. Фрагмент файла DierollCtl.cpp — настройка активизации при появлении на экране

```
/////////////////////////////////////////////////////////////////////////////
// Control type information
// Информация о типе элемента управления.

static const DWORD BASED_CODE _dwDierollOleMisc =
    OLEMISC_ACTIVATEWHENVISIBLE |
    OLEMISC_SETCLIENTSITEFIRST |
    OLEMISC_INSIDEOUT |
    OLEMISC_CANTLINKINSIDE |
    OLEMISC_RECOMPOSEONRESIZE;

IMPLEMENT_OLECTLTYPE(CDierollCtrl, IDS_DIEROLL, _dwDierollOleMisc)
```

Удалите константу OLEMISC_ACTIVATEWHENVISIBLE. Откомпилируйте распространяемую версию приложения. Хотя размер файла Dieroll.OCX и не изменился, страницы Web с этим файлом будут загружаться быстрее, поскольку до первого щелчка на игральной кости окно не создается. Если вы загрузите страницу Web с игральной костью, то сразу же увидите первое

значение, несмотря на то что элемент управления неактивен. Окно создается для того, чтобы обрабатывать щелчки мышью, а не отображать игральную кость.

Существуют и другие способы оптимизации. На рис. 20.3 показан список дополнительных параметров ActiveX ControlWizard, доступ к которым осуществляется с помощью кнопки Advanced на втором этапе настройки. Все эти параметры можно установить при настройке ControlWizard. Их также можно изменить в существующем приложении, что избавляет от необходимости повторного запуска AppWizard и добавления собственных функций. Ниже приведен список возможных параметров.

- Безоконная активизация (флажок Windowless activation)
- Контекст устройства, игнорирующий отсечение (флажок Unclipped device context)
- Активизация без мигания (флажок Flicker-free activation)
- Извещения о перемещении указателя мыши в неактивном состоянии элемента (флажок Mouse pointer notification when inactive)
- Оптимизированный код перерисовки (флажок Optimized drawing code)
- Асинхронная загрузка свойств (флажок Loads properties asynchronously)

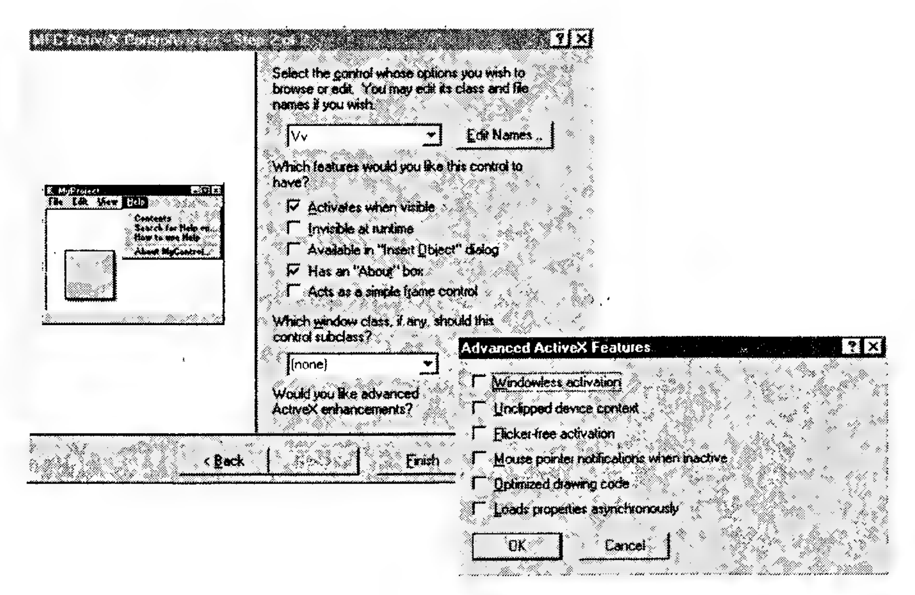


Рис. 20.3. Кнопка Advanced, расположенная в окне ControlWizard, ведет к списку параметров оптимизации

Ввиду своих несомненных достоинств безоконная активизация становится очень популярной. Она используется при необходимости получить прозрачный или непрямоугольный элемент управления. Мы рекомендуем рассмотреть возможность ее использования при разработке любого элемента управления, поскольку она уменьшает размер кода и ускоряет выполнение. Современные контейнеры берут на себя некоторую часть забот о нормальном функционировании элемента управления. В более старых контейнерах окно все равно создается, так что результат оптимизации теряется, но, тем не менее, гарантируется функционирование элемента управления.

Чтобы реализовать безоконную активизацию в Dieroll, перегрузите функцию CDierollCtrl::GetControlFlags() следующим образом:

```
DWORD CDierollCtrl::GetControlFlags()
{
   return COleControl::GetControlFlags() | windowlessActivate;
}
```

Функцию можно добавить быстро, щелкнув правой кнопкой мыши на CDierollCtrl в окне Class View и выбрав из контекстного меню Add Function. После редактирования функции откомпилируйте новую версию Dieroll и загрузите страницу Web с этим элементом управления. Вы не заметите никакого очевидного эффекта от внесенных изменений, поскольку Dieroll и так работает быстро. Однако, по крайней мере, вы сможете убедиться в том, что он будет по-прежнему работать хорошо даже без окна.

Следующие две опции настройки — *контекст устройства, игнорирующий отсечение* (флажок Unclipped device context), и *активизация без мигания* (флажок Flicker-free activation) — недоступны в элементах управления, не имеющих окон. В элементе управления с окном выбор контекста устройства, игнорирующего отсечение, означает, что вы абсолютно уверены в том, что в процессе отрисовки никогда не произойдет выход за пределы прямоугольника, выделенного для элемента управления (прямоугольника клиента). При отсутствии проверки на выход за пределы этого прямоугольника элемент управления работает быстрее, хотя при наличии ошибок в программе отрисовки могут возникнуть проблемы. Если бы это иеобходимо было сделать в Dieroll, переопределенная функция GetControlFlags() выглядела бы так:

```
DWORD CDierollCtrl::GetControlFlags()
{
   return COleControl::GetControlFlags() & ~clipPaintDC;
}
```

Не пытайтесь совместить неотсекающий контекст устройства с безоконной активизацией — это не даст никакого эффекта.

Активизация без мигания полезна для элементов управления, активные и неактивные изображения которых идентичны. (Вспомните главу 15, в которой объект сервера изображался менее яркими цветами, когда объекты были неактивны.) Если перерисовка не нужна, так как программа отрисовки идентична, можно выбрать этот параметр и исключить таким образом вторую перерисовку. Пользователи не увидят раздражающего мигания при активизации элемента управления, а сама активизация будет происходить немного быстрее. Если бы это необходимо было сделать в Dieroll, переопределенная функция GetControlFlags() выглядела бы так:

```
DWORD CDierollCtrl::GetControlFlags()
{
   return COleControl::GetControlFlags() | noFlickerActivate;
}
```

Как и в случае с контекстом устройства, игнорирующим отсечение, комбинирование этого параметра с безоконной активизацией ничего не дает.

Извещения об указателе мыши в неактивном состоянии элемента позволяют для большинства элементов управления выключить опцию Activate When Visible. Если единственной причиной быть активным является необходимость иметь окно, чтобы обрабатывать действия мыши, то при установке этого параметра информацию об этих действиях можно будет получать в контейнере с помощью интерфейса IPointerInactive. Чтобы обеспечить такую возможность в существующем приложении, необходимо снова переопределить GetControlFlags():

```
DWORD CDierollCtrl::GetControlFlags()
{
   return COleControl::GetControlFlags() | pointerInactive;
}
```

Теперь элемент управления будет получать сообщения WM_SETCURSOR и WM_MOUSEMOVE через карту сообщений, даже если у него нет окна. Контейнер, окно которого используется элементом управления, будет посылать ему эти сообщения с помощью интерфейса IPointerInactive.

Еще одна ситуация, в которой может потребоваться обработка оконных сообщений, в то время как окно не активно (т.е. выполнять эти действия, фактически не имея окна), возникает, когда пользователь перетаскивает что-то на элемент управления. В этот момент элемент управления должен активизироваться, чтобы у него появилось окно адресата перетаскивания. Этого можно добиться, переопределив метод GetActivationPolicy():

```
DWORD CDierollCtrl::GetActivationPolicy()
{
  return POINTERINACTIVE_ACTIVATEONDRAG;
}
```

Конечно, если элемент управления не является адресатом перетаскивания, делать это не имеет смысла.

Возложение на контейнер обязанности передавать сообщения с помощью интерфейса IPointerInactive влечет за собой определенные сложности, поскольку контейнер может не знать о существовании такого интерфейса и не планировать передачу с его помощью сообщения. Если элемент управления может оказаться в таком контейнере, не нужно удалять флаг OLEMISC_ACTIVATEWHENVISIBLE из фрагмента программы, представленного выше, в листинге 20.5.

Вместо этого добавьте к флагам с помощью операции побитового ИЛИ еще один — OLEMISC_IGNOREACTIVATEWHENVISIBLE. Этот флаг со странным названием имеет значение для контейнеров, которым известен интерфейс IPointerInactive, и означает *Я забираю это обратно, не активизируйте вообще при появлении*. Контейнеры, которые не распознают интерфейс IPointerInactive, этот флаг также игнорируют. В результате в таких контейнерах элемент управления будет активизирован сразу же после вывода на экран, и поэтому будет готов обрабатывать сообщения.

Оптимизировать программу отрисовки полезно только в элементах управления, которые содержатся в одном контейнере вместе с несколькими другими элементами управления. Как вы помните из главы 5, типичные действия для рисования любого объекта — установить кисть, перо или другой объект GDI, сохранив старое значение, использовать этот объект, а затем восстановить прежнее значение. Если такие действия по очереди выполняют несколько элементов управления, все операции восстановления объектов GDI можно заменить одним общим восстановлением после выполнения всех процедур рисования. Контейнер сохраняет все объекты GDI перед запуском перерисовки всех элементов управления и восстанавливает их после ее окончания.

Чтобы ваш элемент управления воспользовался преимуществом такой организации, необходимо внести два изменения. Во-первых, если перо или другой объект GDI должен оставаться связанным с переменной между вызовами процедуры рисования, эта переменная не должна выходить за пределы области видимости. Это значит, что все локальные перья, кисти и шрифты должны быть преобразованы в члены объекта, чтобы они оставались в области видимости между вызовами функции. Во-вторых, текст программы, восстанавливающий старые объекты, должен быть помещен в оператор if, проверяющий необходимость восстановления обращением к COleControl::IsOptimizedDraw(). Типичная процедура рисования должна установить цвета, а затем продолжать работу следующим образом.

```
...
if(!m_pen.m_hObject)
{
   m_pen.CreatePen(PS_SOLID, 0, forecolor);
}
if(!m_brush.m_hObject)
{
   m_brush.CreateSolidBrush(backcolor);
}
CPen* savepen = pdc->SelectObject(&m_pen);
CBrush* savebrush = pdc->SelectObject(&m_brush);
...
// Использовать контекст устройства.
...
if(!IsOptimizedDraw())
{
   pdc->SelectObject(savepen);
   pdc->SelectObect(savebrush);
}
...
```

В контексте устройства есть адреса членов объекта, поэтому, когда управление передается контейнеру, переменная-член m_hObject становится равной NULL. Пока она не равна NULL, нет необходимости инициализировать контекст устройства, а если контейнер поддерживает оптимизированную перерисовку, также нет необходимости восстанавливать контекст.

Если вы выберете в AppWizard оптимизированную перерисовку, в текст программы добавится оператор if, включающий вызов IsOptimizedDraw(), и несколько комментариев, напоминающих о том, что надо делать.

Последний параметр оптимизации асинхронно загружает свойства; его использование рассматривается в следующем разделе.

Ускорение работы элементов управления с помощью асинхронных свойств

Термин *асинхронный* по отношению к действиям означает, что они могут происходить на протяжении некоторого времени и завершение одного действия до начала другого не обязательно. В контексте Web имеет смысл напомнить о преимуществе Netscape Navigator перед Mosaic, когда он впервые появился на рынке. Первое преимущество, отмеченное пользователями, которые работали с Web, заключалось в том, что броузер Netscape в отличие от Mosaic мог отображать текст в то время, когда графические изображения продолжали загружаться. Это классический пример асинхронного протекания процессов. Не нужно ждать окончания передачи огромных файлов изображений для того, чтобы прочитать слова на странице и решить, стоит ли вообще ждать появления этих изображений на экране.

Повышение скорости подключения к Internet и более компактные форматы изображений уменьшили неудобство ожидания загрузки изображений. Тем не менее асинхронное протекание процессов весьма привлекательно. Из-за долгого ожидания видео- и звуковых клипов и выполняемого кода многие пользователи Web с сожалением вспоминают добрые старые времена, когда для загрузки всех изображений на странице необходимо было ждать всего 30 секунд.

Свойства

Игральная кость на странице Web — изображение кости, принимаемое по умолчанию. Пользователи не имеют возможности получить доступ к свойствам элемента управления. Разработчики страниц Web могут делать это с помощью дескриптора <PARAM> внутри дескриптора <OBJECT>. (Броузеры, не распознающие <OBJECT>, также игнорируют <PARAM>.) Следующий дескриптор <PARAM>, включенный в файл HTML между <OBJECT> и </OBJECT>, вызовет вывод игральной кости с числом, а не точками:

```
<PARAM NAME ="Dots" value="0">
```

У дескриптора <PARAM> есть два атрибута: NAME, задающий имя, соответствующее внешнему имени элемента ActiveX, и value, указывающий значение (в данном примере — 0 или FALSE). В таком случае игральная кость выводится с числом.

Чтобы продемонстрировать значение асинхронных свойств, у объекта Dieroll должны быть какие-то "большие" свойства (в смысле длины соответствующего выполняемого кода). В связи с тем что это демонстрационное приложение, следующим шагом будет добавление "большого" свойства. Естественный выбор — предоставить пользователю больше возможностей управлять видом игральной кости. Пользователь (которым является разработчик страницы Web, если элемент управления применяется в странице Web) может указать имя файла изображения, который будет использоваться в качестве фона кости. Прежде чем продемонстрировать, как это делается, задумайтесь, на каких операциях пользователь страницы Web будет простаивать в ожидании завершения процесса при загрузке страницы с элементом управления Dieroll.

- С сервера загружается текст HTML.
- Броузер размещает текстовые и нетекстовые элементы и начинает вывод текста.
- Броузер ищет CLSID элемента управления в реестре.
- Если необходимо, на основании параметра CODEBASE загружается элемент управления.
- С помощью дескрипторов PARAM инициализируются свойства элемента управления.
- Элемент управления запускается и выводит свое изображение.

Когда в Dieroll загружается еще одно свойство — файл изображения, размер которого может быть большим, — еще одна задержка возникнет при загрузке файла из места его хранения. Если в это время ничего не будет происходить, читателю страницы Web может надоесть смотреть на пустой квадрат, и он перейдет на другую страницу. Использование асинхронных свойств означает, что элемент управления может грубо нарисовать себя и сразу же стать видимым, в то время как большой файл изображения еще загружается. В случае Dieroll подойдет изображение точек на пустом фоне с помощью GetBackColor() до момента готовности файла изображения.

Использование BLOB

BLOB — это большой двоичный объект (Binary Large OBject). Этим общим именем называются такие программные объекты, как файлы изображений наподобие того, который мы хотим добавить в элемент управления Dieroll. Элемент управления общается с BLOB с помощью специального программного инструмента — *моникера* (moniker). В этом нет ничего нового, просто моникеры всегда были скрыты внутри OLE. Если вы с ними уже знакомы, вам придется еще многое узнать о них, потому что с введением асинхронных моникеров все меняется. Если вы никогда о них не слышали, ничего страшного. Со временем появятся все виды асинхронных моникеров, но сейчас реализованы только моникеры адресов. Есть способы

связать свойства BLOB с адресами в ActiveX. Если вы готовы доверить ActiveX сделать это вместо вас, вы достигнете удивительных результатов. В оставшейся части этого подраздела рассказывается, как работать с моникерами адресов, чтобы асинхронно загружать свойства BLOB.

Напомним — идея заключается в том, что элемент управления начинает отображать себя еще до того, как все свойства станут доступными. Функция OnDraw() будет иметь следующую структуру.

```
// Подготовиться к рисованию,
if(AllPropertiesAreLoaded) // если все свойства загружены.
{
   // Отображать с использованием BLOB.
}
else
{
   // Отображать без BLOB.
}
// Очистка после рисования.
```

Здесь возникают две проблемы. Во-первых, как проверить, что все свойства загружены? Во-вторых, как можно осуществить вызов OnDraw() второй раз, когда свойства готовы, если она уже была вызвана и отобразила элемент управления "безBLOBным" способом?

Первая проблема была решена с помощью добавления двух новых методов в класс COleControl. Функция GetReadyState() возвращает одно из следующих значений.

- READYSTATE_UNINITIALIZED означает, что элемент управления совсем не инициализирован.
- READYSTATE_LOADING означает, что свойства элемента управления *загружаются*.
- READYSTATE_LOADED означает, что все свойства *уже загружены*.
- READYSTATE_INTERACTIVE означает, что элемент управления может взаимодействовать с пользователем, но еще не загружен полностью.
- READYSTATE_COMPLETE означает, что ждать больше нечего.

Функция InternalSetReadyState() присваивает состоянию готовности одно из этих значений.

Вторая проблема — получение второго вызова OnDraw() после того, как элемент управления был нарисован без BLOB, — была решена с помощью нового класса CDataPathProperty и производного от него CCachedDataPathProperty. В этих классах есть метод OnDataAvailable(), перехватывающий сообщение Windows, которое посылается, когда свойство получено из удаленного источника. Функция OnDataAvailable() вызывает InvalidateControl() для области элемента управления и таким образом инициирует его перерисовку.

Изменение элемента управления Dieroll

Сделайте копию папки Dieroll, которую вы создали в главе 17, и установите безоконную активизацию так, как это было описано ранее в данной главе. Теперь можно приступать. Для реализации асинхронных свойств необходимо многое сделать, но каждый шаг сам по себе довольно прост.

Добавьте класс CDierollDataPathProperty. Откройте ClassWizard, щелкните на вкладке Automation, а затем на кнопке Add Class. Из раскрывающегося меню, которое появится под кнопкой, выберите команду New. После этого откроется окно New Class. Назовите класс CDierollDataPathProperty. Щелкните на кнопке раскрывающегося списка Base class и выберите CCachedDataPathProperty. Диалоговое окно должно выглядеть так, как показано на рис. 20.4. Щелкните на OK, чтобы создать класс и добавить его в проект.

Новый класс должен быть производным от класса CCachedDataPathProperty, поскольку данные свойства будут загружаться в файл (так легче обрабатывать растровое изображение). Если у элемента управления есть свойство, которое загружается из-за частого изменения (например, погода), лучше было бы выбрать в качестве базового CDataPathProperty.

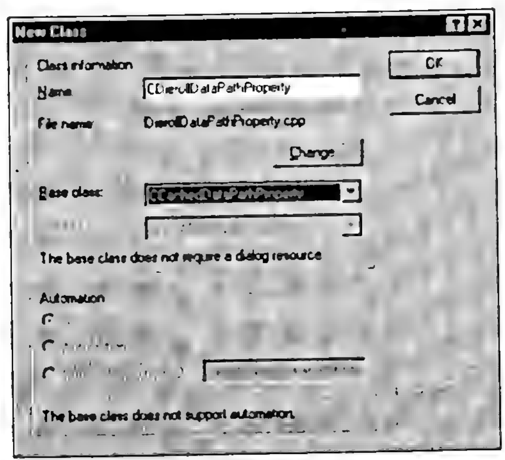


Рис. 20.4. Создание нового класса для обработки асинхронных свойств

Добавьте в класс CDierollCtrl свойство рисунка. После добавления в элемент управления Dieroll класса CDierollDataPathProperty создайте новое свойство в скопированном вами исходном классе CDierollCtrl: в ClassWizard на вкладке Automation выберите в правом раскрывающемся списке CDierollCtrl. Щелкните на кнопке Add Property и заполните диалоговое окно так, как показано на рис. 20.5. Внешнее имя, выбираемое вами, будет фигурировать в тексте HTML: Image является подходящим именем, которое просто набирать. Тип должен быть BSTR — этот пункт не появится в раскрывающемся списке, пока вы не поменяете в группе Implementation выбор опции на Get/Set methods.

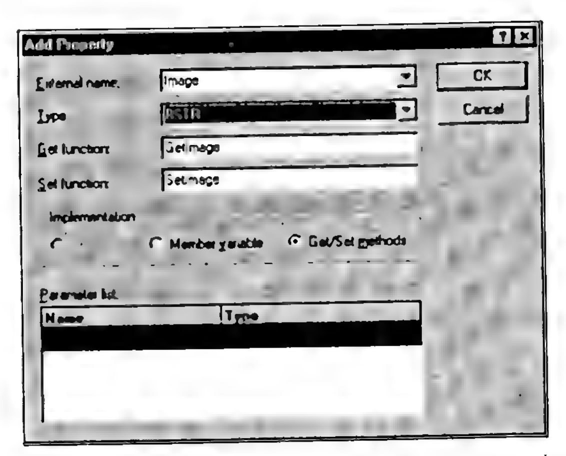


Рис. 20.5. Файл изображения добавляется как свойство типа BSTR

ClassWizard добавляет в класс элемента управления методы Get и Set (листинг 20.6).

Листинг 20.6. Файл DierollCtl.cpp — методы Get и Set

```
BSTR CDierollCtrl::GetImage()
{
   CString strResult;
   // TODO: добавьте здесь обработку свойства.

   return strResult.AllocSysString();
}

void CDierollCtrl::SetImage(LPCTSTR lpszNewValue)
{
   // TODO: добавьте здесь обработку свойства.

   SetModifiedFlag();
}
```

Так же, как и в остальных свойствах Get и Set, необходимо будет добавить переменные-члены класса элемента управления и текст функций, получающих или устанавливающих его значение. Этой переменной-членом будет экземпляр только что созданного класса CDierollDataPathProperty. Щелкните правой кнопкой мыши на CDierollCtrl в окне Class View и выберите из контекстного меню Add Member Variable. На рис. 20.6 показано, как заполнить диалоговое окно, чтобы описать поле mddp_image (символы dpp в имени поля напоминают, что это свойство пути данных (data path property)).

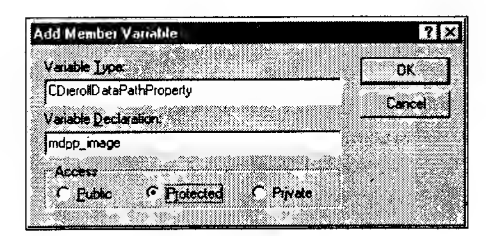


Рис. 20.6. Поле свойства, соответствующее файлу изображения, — экземпляр нового класса

Теперь можно завершить функции Get и Set, как показано в листинге 20.7.

Листинг 20.7. Файл DierollCtl.cpp — окончательные версии функций Get и Set

```
BSTR CDierollCtrl::GetImage()
{
   CString strResult;
   strResult = mdpp_image.GetPath();

   return strResult.AllocSysString();
}

void CDierollCtrl::SetImage(LPCTSTR lpszNewValue)
{
   Load(lpszNewValue, mdpp_image);
   SetModifiedFlag();
}
```

В начале файла заголовка класса CDierollCtrl вставьте следующую строку:

```
#include "DierollDataPathProperty.h"
```

Теперь необходимо слегка изменить несколько функций, поскольку вы вносите изменения в существующий элемент управления, а не включаете асинхронные свойства при создании нового элемента Dieroll. Во-первых, организуйте живучесть и инициализацию mdpp_image, включив в CDierollCtrl::DoPropExchange() следующий оператор:

```
PX_DataPath( pPX, _T("Image"), mdpp_image);
```

Во-вторых, в заготовку функции CDierollCtrl::OnResetState(), созданную ClassWizard, добавьте оператор, восстанавливающий исходное значение свойства пути данных при сбросе элемента управления. Готовая функция представлена в листинге 20.8.

Листинг 20.8. Файл DierollCtl.cpp — CDierollCtrl::OnResetState()

```
////////////////////////////////////////////////////////////////////////////
// CDierollCtrl::OnResetState (Сбрасывает элемент управления в
// состояние по умолчанию.)

void CDierollCtrl::OnResetState()
{
   COleControl::OnResetState();// Восстановить значения
        // по умолчанию в DoPropExchange.
   mdpp_image.ResetData();
}
```

Добавьте событие ReadyStateChange **и свойство** ReadyState. С помощью ClassWizard добавьте событие ReadyStateChange. В окне ClassWizard щелкните на вкладке ActiveX Events, а затем на кнопке Add Event. Выберите ReadyStateChange в раскрывающемся списке и щелкните на OK. На рис. 20.7 показано окно Add Event для этого события. События, как говорилось в главе 17, извещают контейнер элемента управления о том, что что-то произошло в элементе управления. В этом случае таким событием является получение остальных данных и изменение состояния готовности элемента управления.

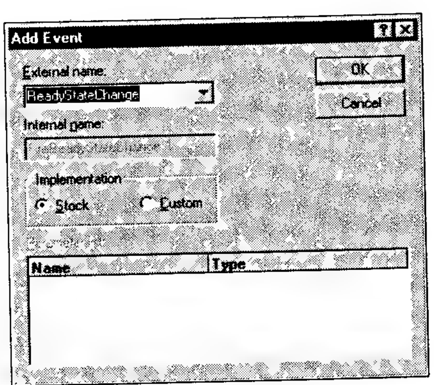


Рис. 20.7. Добавление события, извещающего контейнер об изменении состояния готовности элемента управления

С помощью ClassWizard добавьте в CDierollCtrl свойство состояния готовности. В окне ClassWizard щелкните на вкладке Automation, а затем на кнопке Add Property. Выберите ReadyState в раскрывающемся списке, а поскольку это типовое свойство (stock property), остальная часть диалогового окна заполняется автоматически. ClassWizard не добавляет заготовку функции GetReadyState(), поскольку CDierollCtrl наследует ее от класса COleControl.

Добавьте в конструктор программный код, связывающий кэшируемое свойство с данным элементом управления и инициализирующий переменную-член класса COleControl, которая используется в COleControl::GetReadyState() и устанавливается с помощью COleControl::InternalSetReadyState(). Поскольку элемент управления может быть сразу же использован, состояние готовности должно инициализироваться значением READYSTATE_INTERACTIVE. Новый текст конструктора показан в листинге 20.9.

Листинг 20.9. Файл DierollCtl.cpp — CDierollCtrl::CDierollCtrl()

```
CDierollCtrl::CDierollCtrl()
{
   InitializeIIDs(&IID_DDieroll, &IID_DDierollEvents);
   mdpp_image.SetControl(this);
   m_lReadyState = READYSTATE_INTERACTIVE;
}
```

Реализуйте класс CDierollDataPathProperty. Перед изменением CDierollCtrl::OnDraw() необходимо модифицировать класс CDierollDataPathProperty. Этот класс загружает растровое изображение; в данной главе не объясняется каким образом организуется чтение файла .BMP в объект CBitmap. Самая важная функция — OnDataAvailable(), представленная в листинге 20.10. Добавьте эту функцию — щелкните правой кнопкой мыши на CDierollCtrl в окне Class View и выберите пункт Add Virtual Function в контекстном меню. В левом списке открывшегося диалогового окна выберите OnDataAvailable, затем щелкните на кнопках Add и Edit и наберите текст функции.

Листинг 20.10. Файл DierollDataPathProperty.cpp — OnDataAvailable()

```
void CDierollDataPathProperty::OnDataAvailable(DWORD dwSize, DWORD grfBSCF)
{
   CCachedDataPathProperty::OnDataAvailable(dwSize, grfBSCF);

   if (grfBSCF & BSCF_LASTDATANOTIFICATION)
   {
      m_Cache.SeekToBegin();
      if (ReadBitmap(m_Cache))
      {
         BitmapDataLoaded = TRUE;
         // Безопасно, поскольку у элемента управления
         // только одно свойство:
         GetControl()->InternalSetReadyState( READYSTATE_COMPLETE);
         GetControl()->InvalidateControl();
      }
   }
}
```

Эта функция вызывается всякий раз, когда из удаленного места приходит блок данных. В первой строке используется одноименная функция базового класса, обрабатывающая блок данных и устанавливающая флаг grfBSCF. Если после обработки последнего блока загрузка завершается, вызывается функция ReadBitmap(), считывающая кэшированные данные в объект растрового изображения, который можно вывести в качестве фона элемента управления.

(Текст функции ReadBitmap() здесь не приводится и не обсуждается, хотя для желающих ознакомиться с ним, сообщаем, что он находится на Web-странице и ее можно скопировать в приложение.) После считывания изображения устанавливается окончательное состояние готовности элемента управления, а вызов функции InvalidateControl() обеспечивает его перерисовку.

Измените CDierollCtrl::OnDraw(). Структура функции CDierollCtrl:: OnDraw() была разработана очень давно. Следующий фрагмент текста заполняет фон перед проверкой, что необходимо рисовать — точки или цифры:

```
COLORREF back = TranslateColor(GetBackColor())
CBrush backbrush;
backbrush.CreateSolidBrush(back);
pdc->FillRect(rcBounds, &backbrush)
```

Замените этот фрагмент текстом, приведенным в листинге 20.11.

Листинг 20.11. Файл DierollDataPathProperty.cpp — новый фрагмент OnDraw()

```
CBrush backbrush;
BOOL drawn = FALSE;
if (GetReadyState() == READYSTATE_COMPLETE)
{
   CBitmap* image = mdpp_image.GetBitmap(*pdc);
   if (image)
   {
      CDC memdc;
      memdc.CreateCompatibleDC(pdc);
      memdc.SelectObject(image);
      BITMAP bmp;              //Просто для высоты и ширины.
      image->GetBitmap(&bmp);
      pdc->StretchBlt(0,        //Верхний левый.
            0,        //Верхний правый.
            rcBounds.Width(),  // Необходимая ширина.
            rcBounds.Height(), // Необходимая высота.
            &memdc,    // Изображение.
            0,     // Смещение в изображении -x.
            0,     // Смещение в изображении -y.
            bmp.bmWidth, // Ширина.
            bmp.bmHeight,   // Высота.
            SRCCOPY); //Копировать поверх.

      drawn = TRUE;
   }
}
if (!drawn)
{
   COLORREF back = TranslateColor(GetBackColor());
   backbrush.CreateSolidBrush(back);
   pdc->FillRect(rcBounds, &backbrush);
}
```

Переменная drawn типа BOOL служит для того, чтобы фон был нарисован старым способом, если элемент управления загружен полностью, но использовать растровое изображение по каким-то причинам не удается. Если загрузка элемента управления завершена, изображение загружается в буфер CBitmap*, а затем отображается в контексте устройства. Растровые изображения могут быть выбраны только в контексте устройства памяти, а затем скопированы в обычный контекст устройства. Использование StretchBlt() приводит к масштабированию изображения при копировании, хотя сообразительный разработчик страниц Web установил бы высоту и ширину изображения соответственно атрибутам WIDTH и HEIGHT дескриптора

OBJECT. Старый текст программы рисования по-прежнему присутствует в функции и используется, если значение drawn — FALSE.

Тестирование и отладка элемента управления Dieroll

После внесения всех вышеуказанных изменений откомпилируйте элемент управления, после чего он будет зарегистрирован. Один из способов протестировать его — снова открыть в Explorer страницу HTML, но, скорее всего, вы захотите сначала отладить этот элемент управления. Отладка элементов управления возможна, даже несмотря на то, что их нельзя запустить отдельно. Обычно разработчики помещают отлаживаемый элемент управления в тестовый контейнерный объект, но это необязательно. Для этой цели можно использовать любое приложение, которое может содержать данный элемент управления.

В Visual Studio выберите Project⇨Settings. Щелкните на вкладке Debug и убедитесь в том, что все строки в левом списке выделены. Выберите в раскрывающемся списке пункт General, а в текстовом поле Executable for debugging session введите полный путь к Microsoft Internet Explorer на вашем компьютере (пример показан на рис. 20.8). Теперь при выборе Build⇨Go или после щелчка на кнопке панели инструментов Go будет запускаться Explorer. Откройте страницу HTML, в которой загружается элемент управления, и он будет работать в отладчике. Как и в любом другом приложении, можно устанавливать точки останова, выполнять код пошагово и проверять значения переменных.

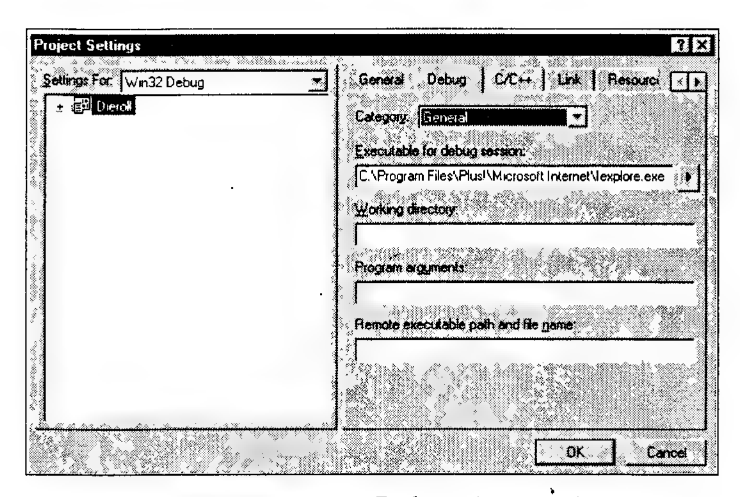


Рис. 20.8. Организуйте запуск Explorer для отладки элемента управления

Свойство Image устанавливается в дескрипторе OBJECT следующим образом.

```
<OBJECT
CLASSID="clsid:46646B43-EA16-11CF-870C-00201801DDD6"
CODEBASE="http://www.gregcons.com/test/dieroll.ocx"
ID=die1
WIDTH=200
HEIGHT=200
ALIGN=center
HSPACE=0
VSPACE=0
>
<PARAM NAME="Dots" VALUE="1">
```

```
<PARAM NAME="Image" VALUE="http://www.gregcons.com/test/beans.bmp">
Если вы видите этот текст, ваш броузер не поддерживает дескриптор OBJECT. </BR>
</OBJECT>
```



Помните, что, если вы самостоятельно создаете Dieroll, недостаточно просто скопировать эти HTML-примеры на свой компьютер. Необходимо использовать собственный CLSID, адрес копии OCX и применяемого файла изображения.

На рис. 20.9 показан элемент управления с фоном из зернышек. Загрузка этой картинки по Web занимает от 30 секунд до одной минуты, и во время загрузки элемент управления прекрасно работает как игральная кость без фона. Это и есть результат применения асинхронных свойств — итог всего проделанного в предыдущем разделе.

Рис. 20.9. Теперь игральную кость можно отобразить на фоне зернышек или любой другой картинки, которую вам подскажет ваша фантазия

ГЛАВА 21

Библиотека Active Template Library

В этой главе...

- Для чего предназначена ATL
- Использование AppWizard на начальной стадии разработки
- Использование Object Wizard
- Добавление свойств элемента управления
- Отображение элемента управления
- Сохранение-восстановление данных и страница свойств
- Использование элемента управления в Control Pad
- Включение событий в программу
- Предоставление функции DoRoll()
- Регистрация элемента управления как безопасного
- Подготовка элемента управления к использованию в режиме разработки
- Минимизация размера выполняемого файла
- Использование элемента управления на странице Web

Библиотека Active Template Library (ATL) — это набор классов C++, которые можно использовать для создания элементов управления ActiveX. В этих небольших элементах управления обычно вообще не используются классы MFC. Для разработки элементов управления ActiveX с помощью ATL требуется более глубокое знание COM и принципов работы интерфейсов, чем для решения аналогичной задачи с использованием MFC. Дело в том, что MFC освобождает программиста от необходимости работать на нижнем уровне интерфейса COM. Использование ATL — не для робких, но отважившиеся пойти этим путем будут вознаграждены, получив элементы управления с меньшим по размеру программным кодом. В этой главе с помощью ATL будет разработан новый вариант элемента управления Dieroll, о котором шла речь в главах 17 и 19. Вы изучите важные понятия COM/ActiveX, которые можно было не принимать во внимание при использовании MFC.

Для чего предназначена ATL

Создать элемент управления ActiveX с помощью MFC довольно просто, как было показано в главах 17 и 20. Для этого не требуется знать, что такое интерфейс COM или как использовать библиотеку типов. Элемент управления может применять всевозможные подходящие классы, такие как CString и CWnd, отображать себя с помощью методов классов CDC и многое другое. Оборотная сторона медали заключается в том, что пользователям нужны библиотеки MFC, и если они не установлены в их системах, загрузка файла CAB размером около 700 Кбайт приведет к значительной задержке.

Альтернативное решение — обеспечивать функции ActiveX с помощью библиотеки Active Template Library (ATL) и вызывать функции SDK Win32 так же, как это делали программисты, разрабатывавшие Windows-приложения до появления Visual C++ и MFC. В данной главе не будет освещена полностью тема SDK Win32 — она слишком обширна. Однако мы можем вас обрадовать — если вы знакомы с большинством классов MFC, таких как CWnd и CDC, то узнаете многие функции SDK, даже если прежде никогда с ними не встречались. Многие методы классов MFC являются просто оболочками вызовов функций SDK.

Сколько же времени загрузки можно сэкономить? Элемент управления на основе MFC из главы 20, занимает около 30 Кбайт, плюс, конечно, библиотеки MFC. Элемент управления на основе ATL, создаваемый в этой главе, занимает самое большее 100 Кбайт, причем в это число входят *все* компоненты кода. С помощью небольших хитростей можно уменьшить его объем до 50 Кбайт плюс 20 Кбайт в библиотеке ATL, т.е. все вместе составляет примерно десятую часть общего объема элемента управления и библиотек из главы 20!

Использование AppWizard на начальной стадии разработки

Для создания элементов управления ATL существует соответствующий мастер. Как всегда, выберите File⇨New и щелкните на вкладке Projects диалогового окна New. Укажите соответствующий каталог и назовите проект DieRollControl (рис. 21.1). Щелкните на OK.

На заметку Не назыеайте проект DieRoll, поскольку позже вы будете вставлять в него элемент управления под названием DieRoll, поэтому во избежание конфликтов имен выберите для проекта более длинное имя.

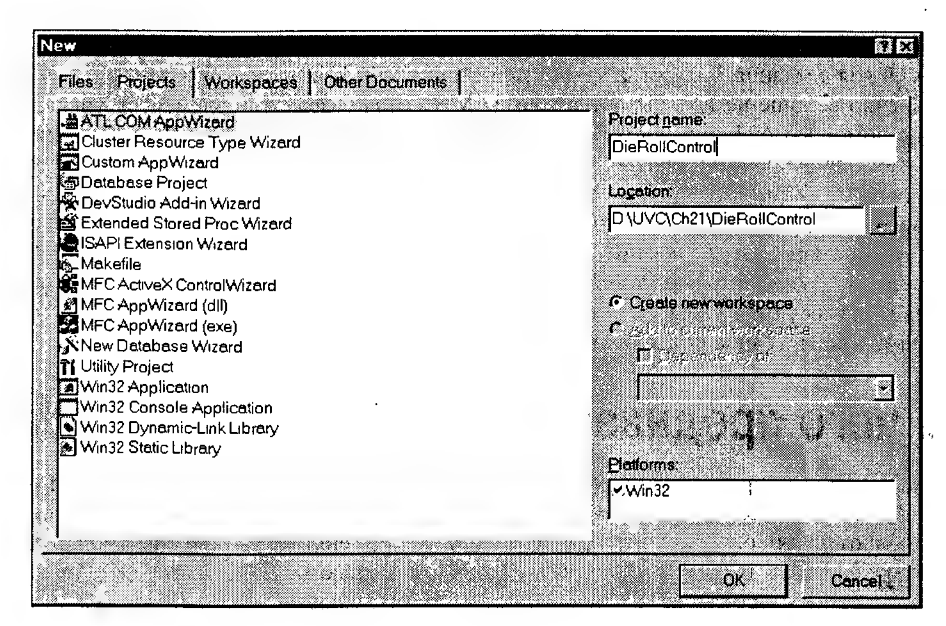


Рис. 21.1. AppWizard упрощает создание элемента управления с помощью ATL

ATL COM Wizard настраивается в один этап (рис. 21.2). Параметры, предлагаемые по умолчанию, — элемент управления DLL (переключатель Dynamic Link Library (DLL)), нет слияния proxy/stub (флажок Allow merging of proxy/stub code сброшен), нет поддержки MFC (флажок Support MFC сброшен) — подходят для данного проекта. Расширение файла будет DLL, а не OCX, как в элементах управления на основе MFC, но это несущественное различие. Щелкните на кнопке Finish.

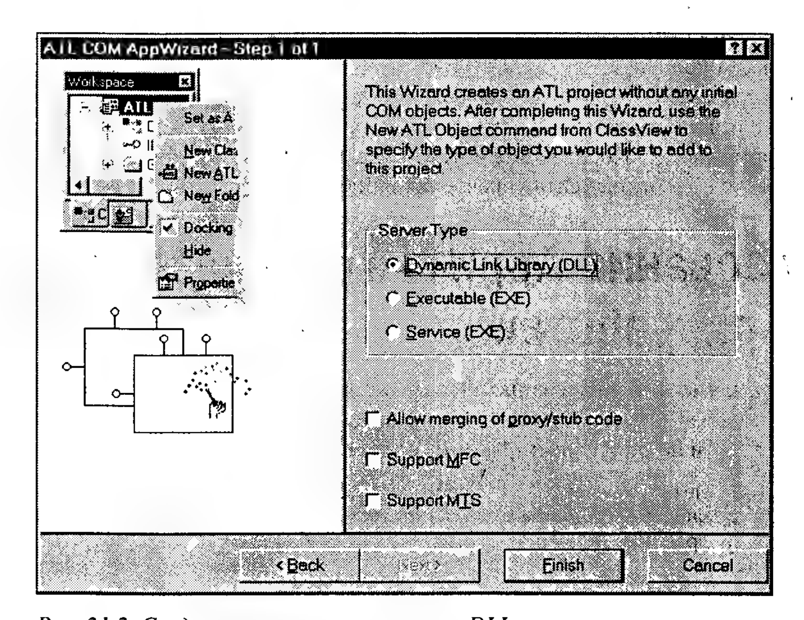


Рис. 21.2. Создание элемента управления DLL.

Инструкция в верхней части окна гласит:

Этот мастер создает ATL-проект, не обращаясь к какому-либо исходному COM-объекту. После завершения настройки мастера выберите команду ATL Object из ClassView, чтобы специфицировать тип объекта, который вы собираетесь добавить в этот проект

Диалоговое окно New Project Information (рис. 21.3) позволяет подтвердить выбранные параметры настройки. Для создания проекта щелкните на кнопке OK.



Рис. 21.3. Сводка выбранных параметров ATL, выводимая перед созданием проекта

Использование Object Wizard

Мастер ATL COM AppWizard создал семь файлов, но заготовка элемента управления еще не готова. Сначала необходимо вставить объект ATL в проект, следуя инструкциям, приведенным в диалоговом окне настройки ATL COM AppWizard - Step 1.

Добавление элемента управления в проект

Выберите в меню Insert⇨New ATL Object. После этого откроется окно ATL Object Wizard (рис. 21.4).

Существует несколько видов объектов ATL, которые можно вставлять в проект, но в данный момент вас интересуют только элементы управления. Поэтому выберите Controls в левом списке. В правом списке предлагается выбор из Full Control (полноценный элемент управления), Lite Control (усеченный элемент управления) и Property Page (страница свойств). Если вы точно знаете, что этот элемент управления будет использоваться только в Internet Explorer, возможно, как часть проекта для intranet, можете выбрать Lite Control и сэкономить немного места. Но данный элемент управления — игральная кость — может ока-

заться в любом броузере, приложении VB или вообще где угодно, и поэтому необходимо выбрать в правом списке Full Control. Позже в этой главе вы займетесь добавлением страницы свойств. Выберите Full Control и щелкните на кнопке Next.

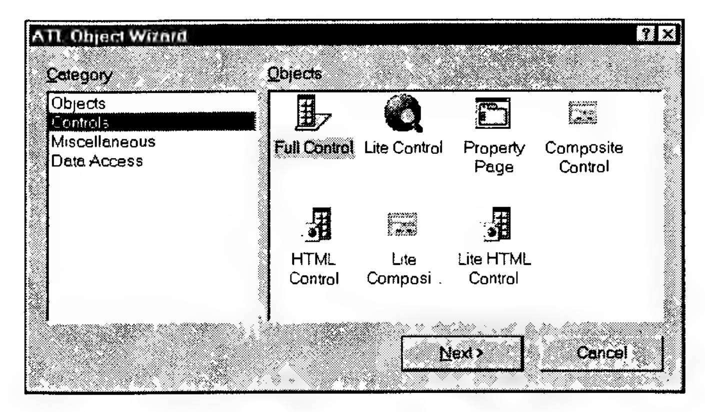


Рис. 21.4. Добавление элемента управления ATL в проект

Выбор имени элемента управления

Следующим появляется диалоговое окно ATL Object Wizard Properties. Его первая вкладка называется Names. Здесь можно указать все имена, используемые в данном элементе управления. В поле Short name (краткое имя) введите DieRoll, а остальные поля заполнятся автоматически на основе введенного имени (рис. 21.5). Имена можно изменить, но в этом нет необходимости. Заметьте, что имя типа (поле Type; DieRoll Class) — это имя, которое будет значиться в диалоговых окнах вставки объекта большинства контейнеров. Поскольку версия Dieroll на основе MFC уже зарегистрирована, данную версию необходимо назвать иначе. В других проектах может понадобиться изменить имя типа.

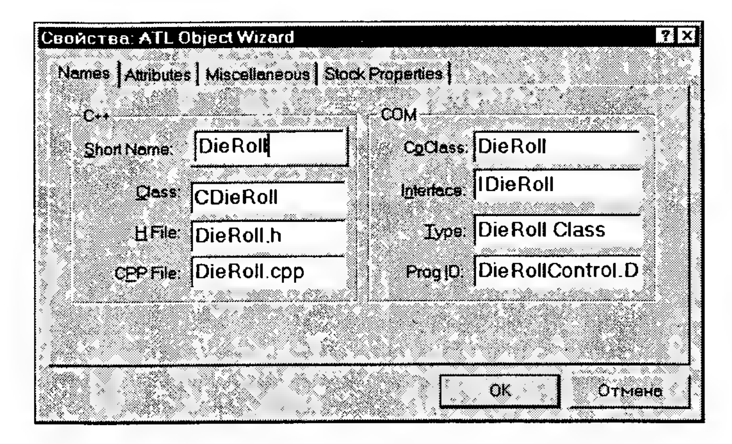


Рис. 21.5. Установка имен файлов и элемента управления

Установка атрибутов элемента управления

Щелкните на вкладке Attributes. Оставьте выбранные по умолчанию опции Apartment в группе Threading Model, Dual — в группе Interface и Yes — в группе Aggregation. Установите флажки Support ISupportErrorInfo и Support Connection Points. Оставьте сброшенным флажок Free Threaded Marshaler (рис. 21.6). Все выбранные параметры рассматриваются ниже.

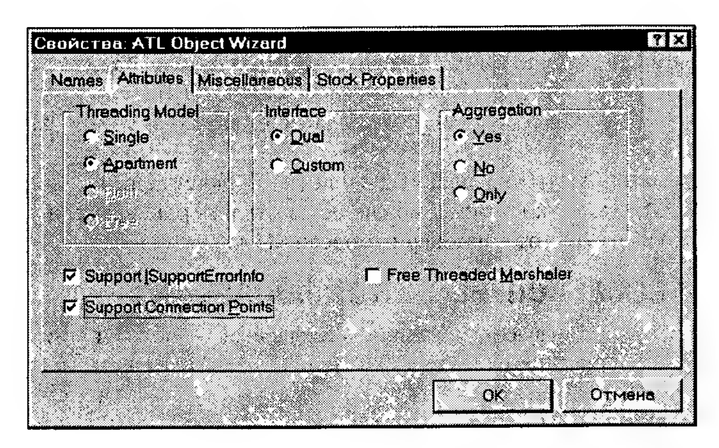


Рис. 21.6. Набор свойств СОМ элемента управления

Модели потока

Не выбирайте модель с единственным потоком (переключатель Single), даже если в элементе управления не используется многозадачность. Чтобы предотвратить одновременное выполнение двух функций такого элемента управления, все вызовы методов элемента управления с единственным потоком должны быть пропущены через proxy-объект, что существенно замедляет выполнение (такая операция называется *маршалингом* (marshaling)). Для вновь создаваемых элементов управления лучше выбирать модель `Apartment`.

Переключатель Apartment устанавливает *однозадачную обособленную модель* функционирования (Single-Threaded Apartment Model — STA). Это значит, что доступ к ресурсам (глобальным и статическим), разделяемым экземплярами элемента управления, осуществляется посредством сериализации. Данным экземпляров объекта — локальным автоматическим переменным и объектам, динамически размещенным в куче, — не нужна защита, которую обеспечивает маршалинг. Вследствие этого элементы управления STA работают быстрее, чем обычные однопотоковые элементы управления. Internet Explorer в своих элементах управления использует модель STA.



Если в процессе проектирования элемента управления используется много глобальных и статических переменных, выбор модели STA потребует от программиста значительного объема работы. Это отнюдь не повод для того, чтобы отказаться от использования модели. Просто нужно "помозговать" над тем, как уменьшить количество глобальных и статических данных. А достичь этого можно единственным способом — применить объектно-ориентированный подход.

Модель потока, которая задается переключателем Free (*многозадачная обособленная модель* — Multithreaded Apartment, MTA), относится к элементам управления, которые являются многозадачными и уже содержат защиту от конфликтов потоков. Хотя разработка многозадачного элемента управления кажется, на первый взгляд, привлекательной, использование та-

кого элемента управления в контейнере, поддерживающем модель с единственным потоком или модель STA, снова приведет к маршалингу, на этот раз, чтобы защитить контейнер от одновременного выполнения двух функций. Это опять же приведет к снижению производительности. Кроме того, программисту, использующему MTA-модель, придется проделать довольно большой объем работы, поскольку нужно будет включить в программу защиту от конфликта потоков.

Выбор опции Both в группе Threading указывает AppWizard на необходимость создания элемента управления, обладающего свойствами и STA, и MTA, что позволяет избежать неэффективного режима выполнения в однопотоковом и STA-контейнерах и полностью использовать возможности модели MTA, если таковая доступна. Примите во внимание, что при этом придется включить в программу защиту от конфликтов, как и при разработке элемеита управления MTA-модели.

В настоящее время элементы управления для Internet Explorer должны быть STA. Элементы управления DCOM, доступ к которым можно осуществлять одновременно по нескольким соединениям, только выиграют от применения модели MTA.

Дуальный и собственный интерфейсы

Объекты COM взаимодействуют с помощью интерфейсов — наборов имен функций, описывающих возможное поведение объекта COM. Чтобы воспользоваться каким-либо интерфейсом, необходимо получить указатель на него, а затем вызвать метод этого интерфейса. У всех серверов автоматизации и элементов управления ActiveX, кроме других интерфейсов, определяющих конкретное назначение сервера или элемента управления, есть интерфейс IDispatch. Чтобы вызвать метод элемента управления, необходимо использовать метод Invoke() интерфейса IDispatch, передав ему идентификатор dispid вызываемого метода. (Этот прием был разработан для того, чтобы методы можно было вызывать из языков, в которых нет указателей, таких как Visual Basic.)

Попросту говоря, элемент управления с *дуальным интерфейсом* (dual-interface) позволяет вызывать методы обоими способами: и посредством функций-членов собственных (custom) интерфейсов, и с помощью интерфейса IDispatch. Элементы управления MFC используют только IDispatch, но этот способ займет больше времени, чем вызов методов собственного интерфейса. В группе Interface можно выбрать опцию Dual или Custom — при выборе опции Custom интерфейс IDispatch вообще не используется, а опция Dual имеет смысл, если предусматривается возможность использования элемента управления в Visual Basic.

Агрегация

Параметры третьей группы, Aggregation, определяют, может ли другой класс COM использовать данный класс COM при том условии, что он будет содержать ссылку на его экземпляр. Yes означает, что другие объекты COM могут использовать этот класс, No — не могут, Only — что они обязаны это делать (данный объект не может работать автономно).

Другие параметры

Установка флажка Support ISupportErrorInfo означает, что элемент управления сможет возвращать контейнеру более полную информацию об ошибке. Установка опции Support Connection Points жизненно необходима для элементов управления, которые генерируют сообщения. Опция Free Threaded Marshaler не имеет смысла в элементах управления модели STA.

Щелкните на вкладке Miscellaneous (Прочие) и просмотрите все опции, значения которых можно оставить без изменения (рис. 21.7). Элемент управления должен быть непрозрачный (флажок Opaque), со сплошным фоном (флажок Solid Background). Он будет использо-

вать нормализованный контекст устройства (флажок Normalized DC), даже несмотря на то, что это слегка снижает эффективность. Дело в том, что программу, отвечающую за вывод на экран, в таком случае будет намного легче разработать.



Если вы хотите знать, как в элементе управления ATL нормализуется контекст устройства, вспомните, что все исходные тексты ATL доступны, так же как и исходные тексты MFC. В файле Program Files\Microsoft Visual Studio\VC98\atl\include\ATLCTL.CPP можно найти функцию CComControlBase::OnDrawAdvanced(), которая нормализует контекст устройства и самостоятельно вызывает OnDraw().

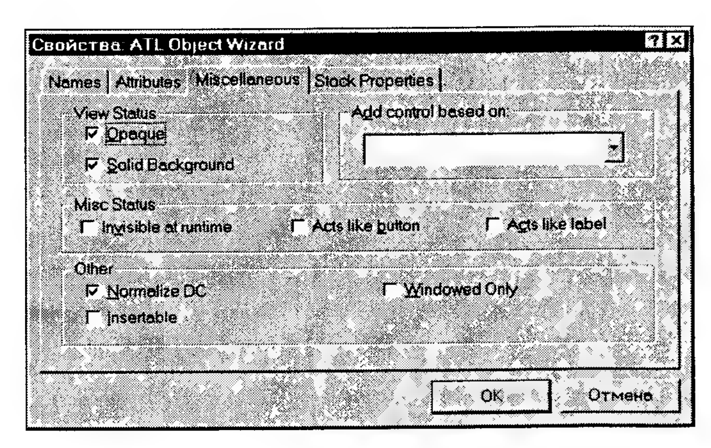


Рис. 21.7. Параметры вкладки Miscellaneous необходимо оставить без изменения

Поддержка типовых свойств

Щелкните на вкладке Stock Properties, чтобы указать, какие типовые свойства будет поддерживать элемент управления. Чтобы добавить поддержку свойства, выберите его в левом списке Not supported, а затем щелкните на кнопке >. Свойство будет перенесено в правый список Supported. Добавьте поддержку Background Color (цвет фона) и Foreground Color (цвет переднего плана), как показано на рис. 21.8. Если вы собираетесь включить поддержку большого количества свойств, воспользуйтесь кнопкой >>, чтобы перенести все свойства из левого списка в список поддерживаемых свойств, а затем перенесите обратно те свойства, поддержку которых считаете излишней.

Щелкните на кнопке OK в окне Object Wizard, чтобы завершить создание элемента управления. На данном этапе можно откомпилировать проект, но элемент управления пока бездействует.

Добавление свойств элемента управления

В версии Dieroll на базе MFC было три типовых свойства: BackColor, ForeColor и ReadyState. Первые два уже добавлены, а свойство ReadyState необходимо добавить вручную. Также есть два пользовательских свойства (Number и Dots) и одно асинхронное (Image).

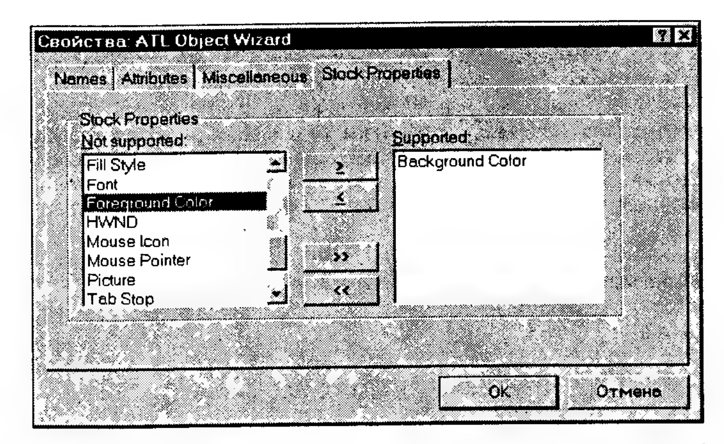


Рис. 21.8. Поддержка свойств Background Color и Foreground Color

Текст программы, генерируемый `Object Wizard`

Реализация или использование интерфейса в классе COM осуществляется путем наследования его от класса, представляющего данный интерфейс. В листинге 21.1 показаны все классы, от которых унаследован `CDieRoll`.

Листинг 21.1. Фрагмент файла `DieRollControl.h` — наследование

```
class ATL_NO_VTABLE CDieRoll :
        public CComObjectRootEx<CComSingleThreadModel>,
        public CComCoClass<CDieRoll, &CLSID_DieRoll>,
        public CComControl<CDieRoll>,
        public CStockPropImpl<CDieRoll, IDieRoll, &IID_IDieRoll,
➥ &LIBID_DIEROLLCONTROLLib>,
        public IProvideClassInfo2Impl<&CLSID_DieRoll, NULL,
➥ &LIBID_DIEROLLCONTROLLib>,
        public IPersistStreamInitImpl<CDieRoll>,
        public IPersistStorageImpl<CDieRoll>,
        public IQuickActivateImpl<CDieRoll>,
        public IOleControlImpl<CDieRoll>,
        public IOleObjectImpl<CDieRoll>,
        public IOleInPlaceActiveObjectImpl<CDieRoll>,
        public IViewObjectExImpl<CDieRoll>,
        public IOleInPlaceObjectWindowlessImpl<CDieRoll>,
        public IDataObjectImpl<CDieRoll>,
        public ISupportErrorInfo,
        public IConnectionPointContainerImpl<CDieRoll>,
        public IPropertyNotifySinkCP<CDieRoll>
        public ISpecifyPropertyPagesImpl<CDieRoll>
```

Теперь вы видите, что обозначает T (template) в названии ATL: все эти классы являются шаблонами классов (template classes). (Если вы не знакомы с шаблонами, прочтите главу 26.) Поддержка другого интерфейса осуществляется с помощью включения еще одной строки в этот список базовых интерфейсных классов.

Ниже в файле заголовка находится карта COM (листинг 21.2).

На заметку

Заметьте, что имена вида IxxxImpl обозначают, что данный класс реализует интерфейс Ixxx. Классы, унаследованные от IxxxImpl, наследуют как имена методов, так и их реализацию. Например, CDieRoll унаследован от ISupportErrorInfo, а не от ISupportErrorInfoImpl<CDieRoll>, несмотря на то, что такой шаблон существует. Это объясняется тем, что текст программы реализации шаблона класса не подходит для элемента управления ATL. Поэтому элемент управления наследует только имена функций исходного интерфейса и, как вы вскоре увидите, должен иметь собственный текст программы реализации в исходном файле.

Листинг 21.2. Фрагмент файла DieRollControl.h — карта COM

```
BEGIN_COM_MAP(CDieRoll)
    COM_INTERFACE_ENTRY_IMPL(IConnectionPointContainer)
    COM_INTERFACE_ENTRY(IDieRoll)
    COM_INTERFACE_ENTRY(IDispatch)
    COM_INTERFACE_ENTRY(IViewObjectEx)
    COM_INTERFACE_ENTRY(IViewObject2)
    COM_INTERFACE_ENTRY(IViewObject)
    COM_INTERFACE_ENTRY(IOleInPlaceObjectWindowless)
    COM_INTERFACE_ENTRY(IOleInPlaceObject)
    COM_INTERFACE_ENTRY(IOleWindow, IOleInPlaceObjectWindowless)
    COM_INTERFACE_ENTRY(IOleInPlaceActiveObject)
    COM_INTERFACE_ENTRY(IOleControl)
    COM_INTERFACE_ENTRY(IOleObject)
    COM_INTERFACE_ENTRY(IPersistStreamInit)
    COM_INTERFACE_ENTRY(IPersist, IPersistStreamInit)
    COM_INTERFACE_ENTRY(ISupportErrorInfo)
    COM_INTERFACE_ENTRY(IConnectionPointContainer)
    COM_INTERFACE_ENTRY(ISpecifyPropertyPages)
    COM_INTERFACE_ENTRY(IQuickActivate)
    COM_INTERFACE_ENTRY(IPersistStorage)
    COM_INTERFACE_ENTRY(IDataObject)
    COM_INTERFACE_ENTRY(IProvideClassInfo)
    COM_INTERFACE_ENTRY(IProvideClassInfo2)

END_COM_MAP()
```

Эта карта связывает IUnknown::QueryInterface() и все интерфейсы, поддерживаемые элементом управления. Все объекты COM должны реализовывать IUnknown, а метод QueryInterface() можно использовать для определения других поддерживаемых интерфейсов и получения указателя на них. Макросы связывают интерфейсы Ixxx с классами IxxxImpl, от которых унаследован CDieRoll.

Возвращаясь к списку наследования класса CDieRoll, можно увидеть, что большая часть шаблонов принимает только один параметр — имя класса — и добавлены AppWizard. Следующий элемент добавлен ObjectWizard:

```
public CStockPropImpl<CDieRoll, IDieRoll, &IID_IDieRoll,
↳ &LIBID_DIEROLLCONTROLLib>,
```

Таким способом Object Wizard осуществляет поддержку типовых свойств. Заметьте, что здесь не указано, какие свойства поддерживаются. Далее в файле заголовка в класс CDieRoll добавлены два члена:

```
OLE_COLOR m_clrBackColor;
OLE_COLOR m_clrForeColor;
```

Object Wizard также модифицирует DieRollControl.idl — файл определения интерфейса (interface definition file), — чтобы показать эти два свойства (листинг 21.3).

Листинг 21.3. Фрагмент файла DieRollControl.idl — типовые свойства

```
[
   object,
   uuid(3767EACE-8CFD-11D0-9B12-0080C81A397C),
   dual,
   helpstring("IDieRoll Interface"),
   pointer_default(unique)
]
interface IDieRoll : IDispatch
{
[propput, id(DISPID_BACKCOLOR)]
   HRESULT BackColor([in]OLE_COLOR clr);
[propget, id(DISPID_BACKCOLOR)]
   HRESULT BackColor([out,retval]OLE_COLOR* pclr);
[propput, id(DISPID_FORECOLOR)]
   HRESULT ForeColor([in]OLE_COLOR clr);
[propget, id(DISPID_FORECOLOR)]
   HRESULT ForeColor([out,retval]OLE_COLOR* pclr);
```

В этом классе осуществляется поддержка функций get и put и извещение контейнера в случае изменения значения одного из свойств.

Добавление типового свойства ReadyState

Несмотря на то что ReadyState не было в списке типовых свойств в ATL Object Wizard, это свойство поддерживается CStockPropImpl. Добавить еще одно свойство можно, отредактировав файлы заголовка и описания интерфейса. В файле заголовка сразу же после строк, описывающих m_clrBackColor и m_clrForeColor, опишите еще одну переменную-член:

```
long m_nReadyState;
```

Это свойство будет использоваться так же, как и свойство ReadyState в версии DieRoll на основе MFC — для реализации Image как асинхронного свойства. В файле DieRollControl.idl добавьте в блок IDispatch после строк для BackColor и ForeColor следующие строки:

```
[propget, id(DISPID_READYSTATE)]
HRESULT ReadyState([out,retval]long* pclr);
```

Добавлять пару строк для реализации функции put этого свойства не нужно, поскольку внешние объекты не могут модифицировать свойство ReadyState. Сохраните файлы заголовка и описания интерфейса, чтобы обновить окно ClassView — если этого не сделать, невозможно будет добавить другие свойства с его помощью. Раскройте в ClassView CDieRoll и IDieRoll, чтобы убедиться, что в CDieRoll были добавлены переменные, а в IDieRoll — функция ReadyState().

Добавление пользовательских свойств

Чтобы добавить собственное свойство, используется инструмент ATL, похожий на ClassWizard MFC. Щелкните правой кнопкой мыши на IDieRoll (элементе верхнего уровня, а не подчиненного CDieRoll) в ClassView, чтобы вывести контекстное меню, показанное на рис. 21.9, и выберите команду Add Property.

Появится диалоговое окно Add Property to Interface (рис. 21.10). В списке Property Type выберите short, а в поле Property Name введите

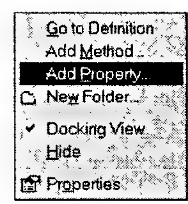


Рис. 21.9. Контекстное меню ClassView в проектах ATL отличается от меню в проектах MFC

Number. Сбросьте флажок Put Function, поскольку контейнерам не понадобится изменять число, показанное на игральной кости. Оставьте остальные параметры без изменения и щелкните на кнопке OK, чтобы добавить свойство.

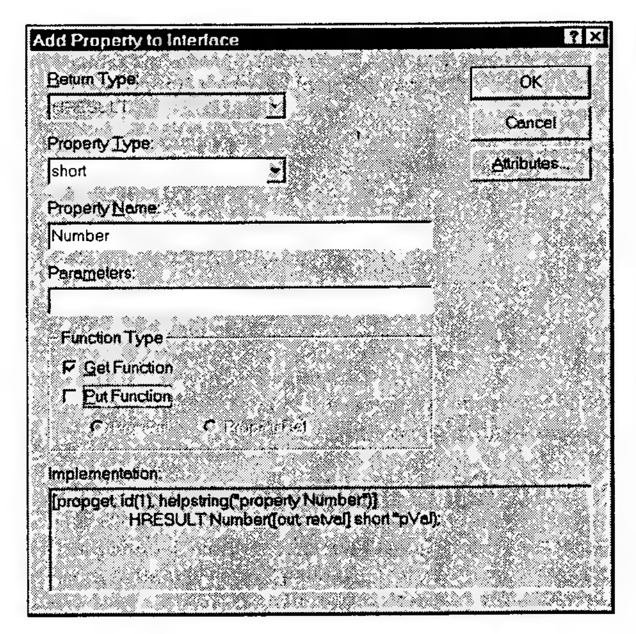


Рис. 21.10. Добавление свойства Number, *доступного только для чтения*

Повторите этот процесс для свойства Dots типа BOOL, у которого должны быть как функция get, так и функция put. (Оставьте установку переключателя PropPut.) Теперь в ClassView появятся элементы, относящиеся к введенным свойствам. Попробуйте дважды щелкнуть на них. Например, при двойном щелчке на get_Dots() в ветви IDieRoll, вложенной в CDieRoll, открывается исходный (.cpp) файл, причем в окне будет выведен текст функции get_Dots(). Двойной щелчок на Dots() в группе IDieRoll верхнего уровня открывает файл описания интерфейса, и в окне будет выведен фрагмент описания propget свойства Dots.

Несмотря на то что в CDieRoll было добавлено несколько элементов, среди них не было ни одной переменной. Только программист может добавлять члены-переменные, соответствующие новым свойствам. Хотя новые свойства с большой вероятностью являются просто членами-переменными класса элемента управления, это может быть и не так. Например, свойство Number могло обозначать размерность некоторого массива, хранящегося в классе, а не переменную.

Добавьте в файл заголовка после описания m_clrBackColor, m_clrForeColor и m_nReadyState следующие строки:

```
short m_sNumber;
BOOL m_bDots;
```

В файле описания интерфейса в новых описаниях propget и propput используются фиксированные dispid, равные 1 и 2:

```
[propget, id(1), helpstring("property Number")]
   HRESULT Number([out, retval] short *pVal);
[propget, id(2), helpstring("property Dots")]
```

```
   HRESULT Dots([out, retval] BOOL *pVal);
[propput, id(2), helpstring("property Dots")]
   HRESULT Dots([in] BOOL newVal);
```

Чтобы сделать текст более читабельным, используйте в качестве типа для dispid перечисление. После включения описания enum в файл описания интерфейса оно будет доступно как в этом файле, так и в файле заголовка. Добавьте в начало файлв DieRollControl.idl следующие строки:

```
typedef enum propertydispids
   {
      dispidNumber = 1,
      dispidDots = 2,
   }PROPERTYDISPIDS;
```

Теперь можно изменить строки описания propget и propput:

```
[propget, id(dispidNumber), helpstring("property Number")]
   HRESULT Number([out, retval] short *pVal);
[propget, id(dispidDots), helpstring("property Dots")]
   HRESULT Dots([out, retval] BOOL *pVal);
[propput, id(dispidDots), helpstring("property Dots")]
   HRESULT Dots([in] BOOL newVal);
```

На следующем этапе необходимо запрограммировать использование переменных-членов в функциях get и set. Окончательный вариант этих функций показан в листинге 21.4. (Если эти функции не видны в окне ClassView, раскройте элемент IDieRoll, вложенный в CDieRoll.)

Листинг 21.4. Фрагмент файла DieRoll.cpp — функции get и set

```
STDMETHODIMP CDieRoll::get_Number(short * pVal)
{
   *pVal = m_sNumber;
   return S_OK;
}

STDMETHDDIMP CDieRoll::get_Dots(BOOL * pVal)
{
   *pVal = m_bDots;
   return S_OK;
}

STDMETHODIMP CDieRoll::put_Dots(BODL newVal)
{
   if (FireOnRequestEdit(dispidDots) == S_FALSE)
   {
      return S_FALSE;
   }
   m_bDots = newVal;
   SetDirty(TRUE);
   FireOnChanged(dispidDots);
   FireViewChange();

   return S_DK;
}
```

Текст двух функций get прост и очевиден. Функция put_Dots() немного сложнее, поскольку она генерирует извещения. FireOnRequestEdit() извещает все интерфейсы IPropertyNotifySink о предстоящем изменении свойства. Любой из этих интерфейсов может

отвергнуть запрос, и, если это происходит, функция возвращает S_FALSE с тем, чтобы запретить изменение.

При условии, что изменение разрешено, переменная-член объекта изменяется, и элемент управления отмечается как измененный ("dirty" — "грязный") с тем, чтобы он был в дальнейшем сохранен. Вызов FireOnChange() уведомляет интерфейсы IPropertyNotifySink об изменении свойства, а вызов FireViewChange() побуждает контейнер перерисовать элемент управления.

Инициализация свойств

После включения фрагментов программы, устанавливающих и возвращающих эти свойства, необходимо изменить конструктор CDieRoll, чтобы инициализировать типовые и пользовательские свойства (листинг 21.5). Заготовка конструктора находится в файле заголовка.

Листинг 21.5. Фрагмент файла DieRoll.h — конструктор

```
CDieRoll()
{
   srand( (unsigned)time( NULL ) );
   m_nReadyState = READYSTATE_INTERACTIVE;
   m_clrBackColor = 0x80000000 | COLOR_WINDOW;
   m_clrForeColor = 0x80000000 | COLOR_WINDOWTEXT;
   m_sNumber = Roll();
   m_bDots = TRUE;
}
```

В начале этого файла вставьте строку, подключающую описание функции time():

```
#include "time.h"
```

Как и в версии этого элемента управления на базе MFC, m_sNumber инициализируется случайным значением от 1 до 6, возвращаемым функцией Roll(). Добавьте в CDieRoll эту функцию, щелкнув правой кнопкой мыши на имени класса в ClassView и выбрав в контекстном меню команду Add Member Function. Roll() является защищенной функцией, не имеет аргументов и возвращает значение типа short. Текст функции Roll() приведен в листинге 21.6. Он был подробно проанализирован в главе 17.

Листинг 21.6. CDieRoll::Roll()

```
short CDieRoll::Roll()
{
   double number = rand();
   number /= RAND_MAX + 1;
   number *= 6;
   return (short)number + 1;
}
```

Теперь имеет смысл оттранслировать проект и убедиться, что в процессе модификации не появились ошибки.

Добавление асинхронного свойства

Как и в главе 20, свойство Image представляет растровое изображение, загружаемое асинхронно и используемое в качестве фона. Добавьте в интерфейс это свойство так же, как и свойства Number и Dots. В качестве типа выберите BSTR, а в качестве имени — Image. Измени-

те описание enum в файле описания интерфейса, чтобы константа dispidImage имела значение 3, и отредактируйте строки propget и propput, используя эту константу:

```
[propget, id(dispidImage), helpstring("property Image")]
   HRESULT Image([out, retval] BSTR *pVal);
[propput, id(dispidImage), helpstring("property Image")]
   HRESULT Image([in] BSTR newVal);
```

Добавьте в класс переменную-член m_bstrImage:

```
CComBSTR m_bstrImage;
```

CComBSTR — это класс ATL, содержащий удобные методы манипулирования BSTR.

Для обработки растрового изображения и асинхронной загрузки необходимо добавить еще несколько членов класса. В файл DieRoll.h вставьте следующие строки:

```
HBITMAP hBitmap;
BITMAPINFOHEADER bmih;
char *lpvBits;
BITMAPINFO *lpbmi;
HGLOBAL hmem1;
HGLOBAL hmem2;
BOOL BitmapDataLoaded;
char *m_Data;
unsigned long m_DataLength;
```

Первые шесть переменных используются для отображения растрового изображения и не будут обсуждаться. Комбинация последних трех обеспечивает такое же поведение свойства пути данных, как и в версии этого элемента управления на базе MFC.

В конструктор добавьте следующие три строки:

```
m_Data = NULL;
m_DataLength = 0;
BitmapDataLoaded = FALSE;
```

Добавьте в файл заголовка класса CDieRoll деструктор, текст которого показан в листинге 21.7.

Листинг 21.7. CDieRoll::~CDieRoll()

```
~CDieRoll()
{
   if (BitmapDataLoaded)
   {
      GlobalUnlock(hmem1);
      GlobalFree(hmem1);
      GlobalUnlock(hmem2);
      GlobalFree(hmem2);
      BitmapDataLoaded = FALSE;
   }

   if (m_Data != NULL)
   {
      delete m_Data;
   }
}
```

У свойства Image есть функции get и put, текст которых приведен в листинге 21.8.

Листинг 21.8. DieRoll.cpp — get_Image() и put_Image()

```
STDMETHODIMP CDieRoll::get_Image(BSTR * pVal)
{
   *pVal = m_bstrImage.Copy();
   return S_OK;
}

STDMETHODIMP CDieRoll::put_Image(BSTR newVal)
{
   USES_CONVERSION;

   if (FireOnRequestEdit(dispidImage) == S_FALSE)
   {
      return S_FALSE;
   }

   // Если было прежнее изображение или данные, удалить их.
      if (BitmapDataLoaded)
   {
      GlobalUnlock(hmem1);
      GlobalFree(hmem1);
      GlobalUnlock(hmem2);
      GlobalFree(hmem2);
      BitmapDataLoaded = FALSE;
   }

   if (m_Data != NULL)
   {
      delete m_Data;
   }

   m_Data = NULL;
   m_DataLength = 0;

      m_bstrImage = newVal;
   LPSTR string = W2A(m_bstrImage);

      if (string != NULL && strlen(string) > 0)
      {
         // Непустая строка, поэтому попытаться загрузить ее.
         BOOL relativeURL = FALSE;
         if (strchr(string, :) == NULL)
            {
                  relativeURL = TRUE;
            }

         m_nReadyState = READYSTATE_LOADING;
            Fire_ReadyStateChange();

         HRESULT ret = CBindStatusCallback<CDieRoll>::Download(this,
               OnData, m_bstrImage, m_spClientSite, relativeURL);
      }
      else
      {
       // Пустая строка; не загружать ее.
         m_nReadyState = READYSTATE_INTERACTIVE;
            Fire_ReadyStateChange();
      }

   SetDirty(TRUE);
   FireOnChanged(dispidImage);
   return S_OK;
}
```

Как и в свойствах Number и Dots, функция get тривиальна, а функцию put рассмотрим подробнее. Ее начало и конец похожи на put_Dots(), и в них также генерируются уведомления для проверки возможности изменения переменной при ее изменении. Код в середине функции относится к данному свойству.

Чтобы начать загрузку свойства, функция вызывает CBindStatusCallback<CDieRoll>::Download(), но сначала необходимо определить, является ли адрес URL m_bstrImage абсолютным или относительным адресом. С помощью макроса ATL W2A строка в формате BSTR преобразуется в обычную строку C, чтобы с помощью стандартной функции strchr() можно было отыскать в адресе символ двоеточия (:). Адрес без двоеточия будет считаться относительным адресом.

На заметку На всех 32-битовых платформах Windows тип BSTR — строка из широких (2 байт) символов. В PowerMac это строка из узких (1 байт) символов.

В версии Dieroll на базе MFC с асинхронной загрузкой свойств всякий раз, когда приходил блок данных, вызывалась функция OnDataAvailable(). После функции Download() при получении данных будет вызываться функция OnData(). Эту функцию необходимо разработать. Добавьте ее в класс вместе с другими открытыми функциями, и реализуйте ее так, как показано в листинге 21.9.

Листинг 21.9. Файл DieRoll.cpp — CDieRoll::OnData()

```
void CDieRoll::OnData(CBindStatusCallback<CDieRoll>* pbsc,
                 BYTE * pBytes, DWORD dwSize)
{
   char *newData = new char[m_DataLength + dwSize];

   memcpy(newData, m_Data, m_DataLength);
   memcpy(newData+m_DataLength, pBytes, dwSize);
   m_DataLength += dwSize;

   delete m_Data;
   m_Data = newData;

   if (ReadBitmap())
   {
      m_nReadyState = READYSTATE_COMPLETE;
   }
}
```

Поскольку при использовании оператора new функция realloc() недоступна, в данном случае используется оператор new для выделения объема памяти, достаточного для хранения уже считанных данных (m_DataLength) и только что полученных данных (dwSize). Затем происходит копирование m_Data и новых данных (размер массива — pBytes байт) в новый блок памяти. После этого происходит попытка преобразовать полученные к этому моменту данные в растровое изображение. Если попытка удается, загрузка считается завершенной и вызовом FireViewChange() контейнеру посылается извещение о необходимости перерисовки элемента управления. Функцию ReadBitmap() можно подключить к проекту, скопировав ее с Web-страницы. Она во многом похожа на версию, использующую MFC, но в ней не применяются классы MFC типа CFile.

Отображение элемента управления

После добавления всех свойств можно перейти к разработке функции OnDraw(). Базовая структура этой функции остается такой же, как и в версии на основе MFC из главы 20, но необходимо выполнить намного больше работы, поскольку невозможно опираться только на MFC.

Структура функции OnDraw() такова:

```
HRESULT CDieRoll::OnDraw(ATL_DRAWINFO& di)
// Если растровое изображение готово, отобразить его.
// Иначе - с помощью BackColor нарисовать сплошной фон.
// Если !Dots, вывести число цветом ForeColor.
// Иначе - нарисовать точки.
```

Сначала необходимо проверить, готово ли растровое изображение, и вывести его, если это возможно. Текст программы приведен в листинге 21.10. Добавьте его к существующей функции OnDraw() вместо текста, оставленного AppWizard. Заметьте, что, если значение ReadyState равно READYSTATE_COMPLETE, но вызов CreateDIBitmap() не возвращает действительный дескриптор растрового изображения, переменные-члены, предназначенные для работы с изображением, очищаются, чтобы дальнейшие вызовы этой функции происходили немного быстрее. В данной главе не обсуждается, как выводить растровые изображения.

Листинг 21.10. CDieRoll::OnDraw() — использование растрового изображения

```
int width = (di.prcBounds->right - di.prcBounds->left + 1);
int height = (di.prcBounds->bottom - di.prcBounds->top + 1);

BOOL drawn = FALSE;
if (m_nReadyState == READYSTATE_COMPLETE)
{
   if (BitmapDataLoaded)
   {
      hBitmap = ::CreateDIBitmap(di.hdcDraw, &bmih, CBM_INIT, lpvBits,
         lpbmi, DIB_RGB_COLORS);

      if (hBitmap)
      {
         HDC hmemdc;
         hmemdc = ::CreateCompatibleDC(di.hdcDraw);
         ::SelectObject(hmemdc, hBitmap);
         DIBSECTION ds;
         ::GetObject(hBitmap,sizeof(DIBSECTION),(LPSTR)&ds);
         ::StretchBlt(di.hdcDraw,
               di.prcBounds->left, // Левый край.
               di.prcBounds->top,  // Верхний край.
               width, // Необходимая ширина.
               height, // Необходимая высота.
               hmemdc, // Изображение.
               0,      //Смещение в изображении -x.
               0,      //Смещение в изображении -y.
               ds.dsBm.bmWidth, // Ширина.
               ds.dsBm.bmHeight, // Высота.
               SRCCOPY);   //Копировать поверх.

         drawn = TRUE;
         ::DeleteObject(hBitmap);
         hBitmap = NULL;
         ::DeleteDC(hmemdc);
      }
```

```
        else
        {
            GlobalUnlock(hmem1);
            GlobalFree(hmem1);
            GlobalUnlock(hmem2);
            GlobalFree(hmem2);
            BitmapDataLoaded = FALSE;
        }
    }
}
```

Если изображение не было выведено из-за того, что ReadyState не равно READYSTATE_COMPLETE или возникла проблема с растровым изображением, OnDraw() рисует сплошной фон с использованием свойства BackColor (листинг 21.11). Добавьте этот текст в OnDraw(). Вызовы функций SDK очень похожи на вызовы соответствующих функций MFC в версии Dieroll на базе MFC. Например, ::OleTranslateColor() соответствует TranslateColor().

Листинг 21.11. CDieRoll::OnDraw() **— рисование сплошного фона**

```
if (!drawn)
{
    COLORREF back;
    ::OleTranslateColor(m_clrBackColor, NULL, &back);
    HBRUSH backbrush = ::CreateSolidBrush(back);
    ::FillRect(di.hdcDraw, (RECT *)di.prcBounds, backbrush);
    ::DeleteObject(backbrush);
}
```

После отображения фона в виде растрового изображения или сплошного фона OnDraw() переходит к обработке переднего плана. Получить цвет переднего плана можно с помощью следующих операторов:

```
COLORREF fore;
::OleTranslateColor(m_clrForeColor, NULL, &fore);
```

Если свойство Dots равно FALSE, игральную кость необходимо отображать с числовым индикатором. Добавьте в OnDaw() текст, приведенный в листинге 21.12, перед оператором return. Функции SDK снова выполняют ту же работу, что и одноименные функции MFC в предыдущей версии Dieroll.

Листинг 21.12. CDieRoll::OnDraw() **— отображение числового индикатора**

```
if (!m_bDots)
{
    _TCHAR val[20]; // Символьное представление значения short.
                    _itot(m_sNumber, val, 10);
        ::SetTextColor(di.hdcDraw, fore);
    ::ExtTextOut(di.hdcDraw, 0, 0, ETO_OPAQUE,
        (RECT *)di.prcBounds, val, _tcslen(val), NULL );
}
```

Текст программы, рисующей точки, приведен в листинге 21.13. Включите его в OnDraw() и завершите разработку функции. Этот фрагмент достаточно велик по объему, но он был проанализирован в главе 17. Как и в остальной части OnDraw(), функции MFC были заменены вызовами SDK.

Листинг 21.13. CDieRoll::OnDraw() — **рисование точек**

```
else
{
   //Ширина и высота точек - 4 единицы, они
   //отстоят от края на одну единицу.
   int Xunit = width/16;
   int Yunit = height/16;
   int Xleft = width%16;
   int Yleft = height%16;

   // Откорректировать верхний левый угол
   // в соответствии с остатком.
   int Top = di.prcBounds->top + Yleft/2;
   int Left = di.prcBounds->left + Xleft/2;

   HBRUSH forebrush;
   forebrush = ::CreateSolidBrush(fore);

   HBRUSH savebrush = (HBRUSH)::SelectObject(di.hdcDraw, forebrush);

   switch(m_sNumber)
   {
   case 1:
       ::Ellipse(di.hdcDraw, Left+6*Xunit, Top+6*Yunit,
          Left+10*Xunit, Top + 10*Yunit); //Средняя.
      break;
   case 2:
      ::Ellipse(di.hdcDraw, Left+Xunit, Top+Yunit,
         Left+5*Xunit, Top + 5*Yunit);   //Верхняя левая.
      ::Ellipse(di.hdcDraw, Left+11*Xunit, Top+11*Yunit,
         Left+15*Xunit, Top + 15*Yunit); //Нижняя правая.
      break;
   case 3:
      ::Ellipse(di.hdcDraw, Left+Xunit, Top+Yunit,
         Left+5*Xunit, Top + 5*Yunit);   //Верхняя левая.
      ::Ellipse(di.hdcDraw, Left+6*Xunit, Top+6*Yunit,
         Left+10*Xunit, Top + 10*Yunit); //Средняя.
      ::Ellipse(di.hdcDraw, Left+11*Xunit, Top+11*Yunit,
         Left+15*Xunit, Top + 15*Yunit); //Нижняя правая.
      break;
   case 4:
      ::Ellipse(di.hdcDraw, Left+Xunit, Top+Yunit,
         Left+5*Xunit, Top + 5*Yunit);   //Верхняя левая.
      ::Ellipse(di.hdcDraw, Left+11*Xunit, Top+Yunit,
         Left+15*Xunit, Top + 5*Yunit);  //Верхняя правая.
      ::Ellipse(di.hdcDraw, Left+Xunit, Top+11*Yunit,
         Left+5*Xunit, Top + 15*Yunit);  //Нижняя левая.
      ::Ellipse(di.hdcDraw, Left+11*Xunit, Top+11*Yunit,
         Left+15*Xunit, Top + 15*Yunit); //Нижняя правая.
      break;
   case 5:
      ::Ellipse(di.hdcDraw, Left+Xunit, Top+Yunit,
         Left+5*Xunit, Top + 5*Yunit);   //Верхняя левая.
      ::Ellipse(di.hdcDraw, Left+11*Xunit, Top+Yunit,
         Left+15*Xunit, Top + 5*Yunit);  //Верхняя правая.
      ::Ellipse(di.hdcDraw, Left+6*Xunit, Top+6*Yunit,
         Left+10*Xunit, Top + 10*Yunit); //Средняя.
      ::Ellipse(di.hdcDraw, Left+Xunit, Top+11*Yunit,
         Left+5*Xunit, Top + 15*Yunit);  //Нижняя левая.
      ::Ellipse(di.hdcDraw, Left+11*Xunit, Top+11*Yunit,
         Left+15*Xunit, Top + 15*Yunit); //Нижняя правая.
```

```
            break;
        case 6:
            ::Ellipse(di.hdcDraw, Left+Xunit, Top+Yunit,
                Left+5*Xunit, Top + 5*Yunit);  //Верхняя левая.
            ::Ellipse(di.hdcDraw, Left+11*Xunit, Top+Yunit,
                Left+15*Xunit, Top + 5*Yunit); //Верхняя правая.
            ::Ellipse(di.hdcDraw, Left+Xunit, Top+6*Yunit,
                Left+5*Xunit, Top + 10*Yunit); //Средняя левая.
            ::Ellipse(di.hdcDraw, Left+11*Xunit, Top+6*Yunit,
                  Left+15*Xunit, Top + 10*Yunit); //Средняя правая.
            ::Ellipse(di.hdcDraw, Left+Xunit, Top+11*Yunit,
                Left+5*Xunit, Top + 15*Yunit); //Нижняя левая.
            ::Ellipse(di.hdcDraw, Left+11*Xunit, Top+11*Yunit,
                Left+15*Xunit, Top + 15*Yunit); //Нижняя правая.
            break;
        }

        ::SelectObject(di.hdcDraw, savebrush);
        ::DeleteObject(forebrush);
    }
    return S_OK;
}
```

Снова оттранслируйте проект, чтобы убедиться, что ничего не забыто во время исправлений. Если теперь вы заглянете в папку проекта, то обнаружите там файл DieRoll.htm, который не появляется в списке окна FileView. Этот HTML-файл генерируется с тем, чтобы можно было протестировать созданный элемент управления. Попробуйте загрузить его в Internet Explorer, и на экране должно появиться изображение игральной кости. Пока что это изображение не имеет фона и никак не реагирует на щелчки мышью.

Сохранение-восстановление данных и страница свойств

В программу элемента управления были добавлены свойства, использующиеся при отображении элемента управления. Теперь остается позаботиться о сохранении-восстановлении соответствующих данных и включить в программу страницу настройки свойств.

Включение страницы свойств

Выберите в меню Insert⇨New ATL Object, чтобы открыть ATL Object Wizard. Выберите в левом списке Controls, в правом списке — Property Page и щелкните на кнопке Next. На вкладке Names введите в поле Short Name имя DieRollPPG, а затем щелкните на вкладке Strings (опции вкладки Attributes останутся без изменения). Введите General в поле Title и DieRoll Property Page — в поле Doc String. Очистите поле Helpfile Name. Щелкните на кнопке OK, чтобы добавить в проект страницу свойств.

Переключитесь на режим ResourceView в левой части экрана среды разработки и откройте диалоговое окно IDD_DIEROLLPPG. Добавьте флажок с идентификатором ресурса IDC_DOTS и этикеткой Display Dot Pattern и текстовое поле с идентификатором ресурса IDC_IMAGE и этикеткой Image URL (рис. 21.11).

В начале файла DieRollPPG.h вставьте следующую строку:

```
#include "DieRollControl.h"
```

Необходимо связать элементы управления проектируемой страницы со свойствами элемента управления DieRoll. Сначала добавьте в карту сообщений в файле DieRollPPG.h три строки, чтобы она приобрела вид, представленный в листинге 21.14.

Листинг 21.14. Файл DieRollPPG.h — карта сообщений

```
BEGIN_MSG_MAP(CDieRollPPG)
   MESSAGE_HANDLER(WM_INITDIALOG, OnInitDialog)
   COMMAND_HANDLER(IDC_DOTS, BN_CLICKED, OnDotsChanged)
   COMMAND_HANDLER(IDC_IMAGE, EN_CHANGE, OnImageChanged)
   CHAIN_MSG_MAP(IPropertyPageImpl<CDieRollPPG>)
END_MSG_MAP()
```

Они предназначены для того, чтобы при инициализации диалогового окна вызывалась функция OnInitDialog(), а при изменении Dots или Image — OnDotsChanged() либо OnImageChanged() (у остальных свойств нет методов put, поэтому они не меняются).

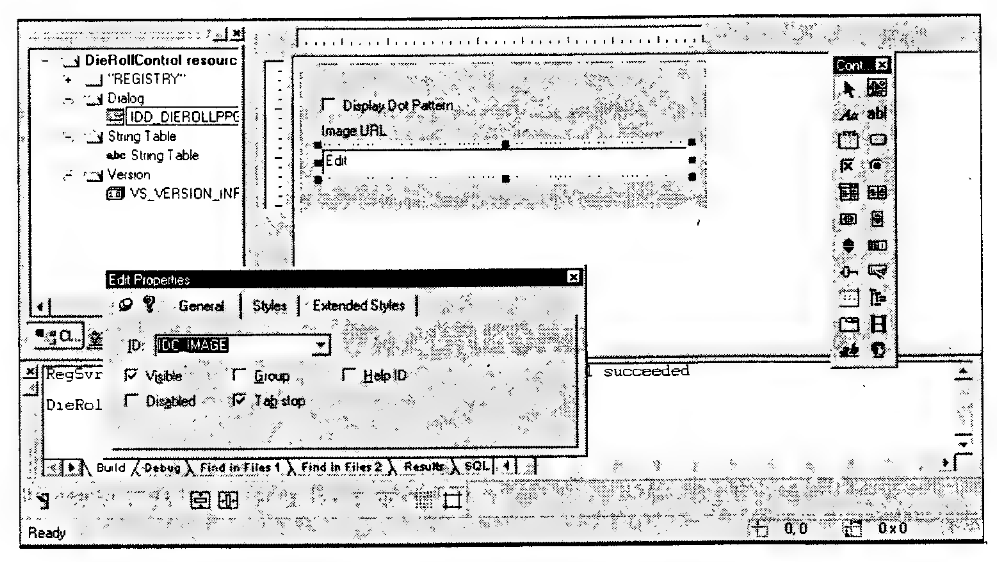


Рис. 21.11. Добавление двух элементов управления в страницу свойств

Добавьте в файл заголовка текст листинга 21.15, чтобы описать и реализовать функцию OnInitDialog().

Листинг 21.15. Файл DieRollPPG.h — CDieRollPPG::OnInitDialog()

```
LRESULT OnInitDialog(UINT uMsg, WPARAM wParam, LPARAM lParam,
                     BOOL & bHandled)
{
   USES_CONVERSION;

   CComQIPtr<IDieRoll, &IID_IDieRoll> pDieRoll(m_ppUnk[0]);

   BOOL dots;
   pDieRoll->get_Dots(&dots);
   ::SendDlgItemMessage(m_hWnd, IDC_DOTS, BM_SETCHECK, dots, 0L);
```

```
    BSTR image;
    pDieRoll->get_Image(&image);
    LPTSTR image_URL = W2T(image);
    SetDlgItemText(IDC_IMAGE, image_URL);

    return TRUE;
}
```

В этой функции сначала с помощью шаблона класса CComQIPtr описывается указатель на интерфейс IDieRoll. Затем этот указатель инициализируется адресом первого элемента массива m_ppUnk, принадлежащего этому же классу CDieRollPPG. (Страница свойств может быть ассоциирована с несколькими элементами управления.) Конструктор класса шаблона CComQIPtr использует метод QueryInterface() переданного конструктору указателя на интерфейс Iunknown с тем, чтобы получить указатель на интерфейс IDieRoll. Теперь можно использовать методы этого интерфейса и получить доступ к свойствам элемента управления DieRoll.

Получить значения свойства Dots объекта DieRoll очень просто — нужно вызвать get_Dots(). Чтобы использовать это значение для инициализации флажка на странице свойств, элементу управления посылается сообщение с помощью функции SDK ::SendDlgItemMessage(). Аргумент BM_SETCHECK указывает, что происходит установка или сброс флажка. Передача dots в качестве четвертого аргумента приведет к установке флажка IDC_DOTS, если dots равно TRUE, и его сбросу, если dots равно FALSE. Аналогично считывается адрес изображения с помощью get_Image(), выполняется его преобразование из формата широких символов и установка содержимого текстового поля с помощью функции SetDlgItemText().

Функции OnDotsChanged() и OnImageChanged() довольно просты — добавьте их текст из листинга 21.16 в файл заголовка после OnInitDialog().

Листинг 21.16. Файл DieRollPPG.h — функции OnChanged

```
LRESULT OnDotsChanged(WORD wNotify, WORD wID, HWND hWnd, BOOL& bHandled)
{
    SetDirty(TRUE);
    return FALSE;
}

LRESULT OnImageChanged(WORD wNotify, WORD wID, HWND hWnd, BOOL& bHandled)
{
    SetDirty(TRUE);
    return FALSE;
}
```

Вызовы SetDirty() в этих функциях приводят к тому, что после щелчка на кнопке OK страницы свойств будет вызвана функция Apply().

Object Wizard сгенерировал простую функцию Apply(), но она не влияет на значения свойств Dots и Number. Отредактируйте эту функцию так, как представлено в листинге 21.17.

Листинг 21.17. Файл DieRollPPG.h — CDieRollPPG::Apply()

```
    STDMETHOD(Apply)(void)
    {
        USES_CONVERSION;
        BSTR image = NULL;
        GetDlgItemText(IDC_IMAGE, image);

        BOOL dots = (BOOL)::SendDlgItemMessage(m_hWnd, IDC_DOTS,
               BM_GETCHECK, 0, 0L);
        ATLTRACE(_T("CDieRollPPG::Apply\n"));
```

```
        for (UINT i = 0; i < m_nObjects; i++)
        {
            CComQIPtr<IDieRoll, &IID_IDieRoll> pDieRoll(m_ppUnk[i]);

            if FAILED(pDieRoll->put_Dots(dots))
            {
                CComPtr<IErrorInfo> pError;
                CComBSTR            strError;
                GetErrorInfo(0, &pError);
                pError->GetDescription(&strError);
                MessageBox(OLE2T(strError), _T("Error"), MB_ICONEXCLAMATION);
                return E_FAIL;
            }

            if FAILED(pDieRoll->put_Image(image))
            {
                CComPtr<IErrorInfo> pError;
                CComBSTR            strError;
                GetErrorInfo(0, &pError);
                pError->GetDescription(&strError);
                MessageBox(OLE2T(strError), _T("Error"), MB_ICONEXCLAMATION);
                return E_FAIL;
            }
        }
        m_bDirty = FALSE;
        return S_OK;
    }
}
```

Операции в функции Apply() начинаются с получения dots и image из диалогового окна. Заметьте, что в вызове ::SendDlgItemMessage() третий аргумент должен быть равен BM_GETCHECK, чтобы этот вызов возвратил состояние флажка (TRUE или FALSE). Затем для облегчения отладки вызывается ATLTRACE и печатает сообщение трассировки. Как и операторы отладки, обсуждаемые в главе 24, этот оператор исчезает при компиляции распространяемой версии программы.

Большую часть функции Apply() занимает цикл for, выполняющийся по одному разу для каждого элемента управления, ассоциированного с данной страницей свойств. В цикле считывается указатель на интерфейс IDieRoll так же, как и в OnInitDialog(), а затем выполняются попытки вызова методов put_Dots() и put_Image() этого интерфейса. Если какой-либо из вызовов завершается неудачей, выводится сообщение об обнаруженной ошибке. После окончания цикла переменной m_bDirty присваивается значение FALSE.

Связывание страницы свойств с CDieRoll

Модификация класса CDieRollPPG завершена. Некоторые изменения необходимо внести в класс CDieRoll, чтобы связать его с классом страницы свойств. А если конкретнее, то необходимо добавить несколько элементов в карту свойств. Отредактируйте ее, чтобы она соответствовала листингу 21.18.

Листинг 21.18. Файл DieRollPPG.h

```
BEGIN_PROPERTY_MAP(CDieRoll)
    PROP_ENTRY( "Dots", dispidDots, CLSID_DieRollPPG)
    PROP_ENTRY( "Image", dispidImage, CLSID_DieRollPPG)
    PROP_ENTRY( "Fore Color", DISPID_FORECOLOR, CLSID_StockColorPage )
    PROP_ENTRY( "Back Color", DISPID_BACKCOLOR, CLSID_StockColorPage )
END_PROPERTY_MAP()
```

Строки для Dots и Image необходимо добавить, а в строках Fore Color и Back Color заменить макрос PROP_PAGE, подставленный Object Wizard, макросом PROP_ENTRY, чтобы свойства были живучими, т.е. сохранялись вместе с элементом управления.

Сохранение-восстановление набора свойств

Существует несколько способов, с помощью которых Internet Explorer может получать значения свойств из текстов HTML элементов управления, заключенных между дескрипторами OBJECT. В потоке сохранения-восстановления, который поддерживается по умолчанию, используется атрибут DATA дескриптора OBJECT. Если вы хотите использовать более читабельные дескрипторы PARAM, элемент управления должен посредством интерфейса IPersistPropertyBag поддерживать сохранение-восстановление набора свойств.

Добавьте еще один класс в список базовых классов в начале описания класса CDieRoll:

```
public IPersistProperyBagImpl<CDieRoll>,
```

Добавьте следующую строку в карту COM:

```
COM_INTERFACE_ENTRY_IMPL(IPersistProperyBag)
```

Теперь для установки свойств элемента управления можно использовать дескрипторы PARAM.

Использование элемента управления в Control Pad

Классы CDieRoll и CDieRollPPG были в значительной мере модифицированы, и настало время откомпилировать элемент управления. После исправления опечаток и мелких ошибок его можно использовать.

Необходимо создать HTML-текст для отображения элемента управления в программе Microsoft Control Pad. Если у вас нет программы Control Pad, ее можно бесплатно получить по адресу **http://www.microsoft.com/workshop/author/cpad/download.htm**. Если ваша копия Control Pad датирована числом до января 1997 года, найдите последнюю версию. При использовании старой версии регистрация безопасной инициализации и обработки сценария будет работать неправильно.

После загрузки Control Pad открывает новый документ HTML. Когда курсор установлен между <BODY> и </BODY>, выберите Edit⇨Insert ActiveX Control. Появится диалоговое окно Insert ActiveX Control. Выберите в списке DieRoll Class (см. рис. 21.5: он напомнит вам, что имя типа данного элемента управления — DieRoll Class) и щелкните на OK. Появится элемент управления и окно Properties. Щелкните на поле свойства Image и введите полный путь к файлу beans.bmp (этот файл можно найти на Web-странице). Щелкните на кнопке Apply, и элемент управления будет отображен с фоном из зернышек (рис. 21.12). Закройте окна Properties и Edit ActiveX Control, и вы увидите автоматически сгенерированный текст HTML, включающий дескрипторы <PARAM>, которые были добавлены, поскольку Control Pad определил, что DieRoll поддерживает интерфейс IPersistPropertyBag.

Сейчас элемент управления еще не работает — после щелчка мышью кость не крутится. В следующем разделе будут добавлены события.

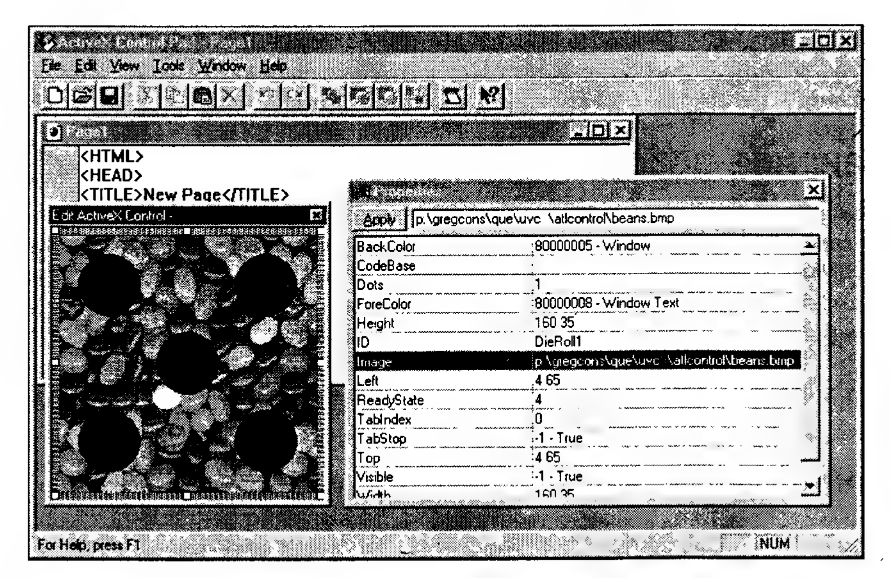


Рис. 21.12. После вставки элемента управления Control Pad *отображает его*

Включение событий в программу

Необходимо включить в программу обработку двух событий — щелчка мышью на элементе управления и изменения состояния готовности. Событие `Click` обсуждалось в главе 17, а событие `ReadyStateChanged` — в главе 20.

Добавление методов в интерфейс событий

В окне ClassView дважды щелкните на названии интерфейса `_IDieRollEvents`. Выберите в контекстном меню пункт Add Method и заполните поле Return Type значением `void`, а поле Method Name — значением `Click`. Поля остальных параметров оставьте пустыми. На рис. 21.13 показано заполненное диалоговое окно. Щелкните на OK — и метод будет добавлен.

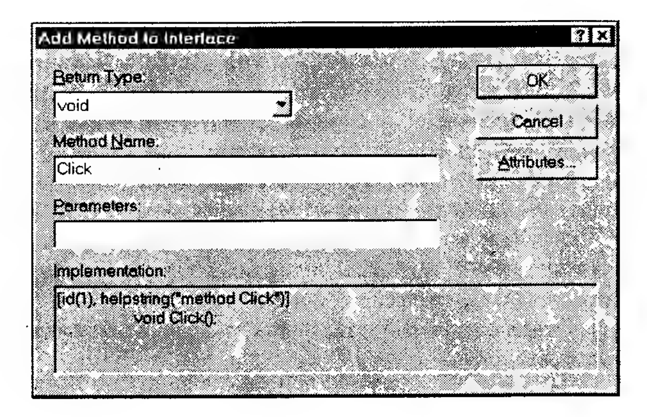


Рис. 21.13. Добавление метода Click() *в интерфейс событий*

Так же добавьте и метод ReadyStateChabnge(), который возвращает тип void и не получает никаких параметров. Секция dispinterface в idl-файле должна выглядеть следующим образом:

```
dispinterface _IDieRollEvents
{
     properties:
     methods:
   [id(DISPID_CLICK)] void Click();
   [id(DISPID_READYSTATECHANGE),
↳ helpstring("method ReadyStateChange")] void ReadyStateChange();
};
```

Реализация интерфейса IConnectionPoint

Чтобы генерировать события, необходимо реализовать интерфейс IConnectionPoint. Мастер Connection Point Wizard поможет вам начать. Прежде всего сохраните файл .idl и откомпилируйте проект, чтобы библиотека типа, ассоциированная с проектом, содержала последнюю информацию.

В окне ClassView дважды щелкните на CDieRoll. Выберите в контекстном меню пункт Implement Connection Point. На экране появится диалоговое окно мастера Connection Point Wizard. Установите флажок возле элемента _IDieRollEvents в поле Interfaces и щелкните на OK — будет сформирован proxy-класс, в котором имеются методы для генерирования событий.

Созданный класс оболочки CProxy_DieRollEvents можно увидеть в окне ClassView. Разверните его, и увидите два метода — Fire_Click() и Fire_ReadyStateChange().

Генерирование события Click

Когда пользователь щелкает мышью на элементе управления, должно генерироваться событие Click. Добавьте в карту сообщений следующую строку:

```
MESSAGE_HANDLER(WM_LBUTTONDOWN, OnLButtonDown)
```

Добавьте в класс CDieRoll метод OnLButtonDown() (листинге 21.19).

Листинг 21.19. CDieRoll::OnLButtonDown()

```
STDMETHODIMP CDieRoll::DoRoll()
{
   m_sNumber = Roll();
   FireOnChanged(dispidNumber);
   FireViewChange();
   return S_OK;
}
```

Эта функция имитирует бросок игральной кости, генерирует событие Click и извещение о том, что значение Number изменилось, а также уведомляет контейнер о том, что элемент управления необходимо перерисовать.

Генерирование события ReadyStateChanged

Функции put_Image() и OnData() теперь могут при изменении состояния готовности генерировать события. Существует два способа дать понять контейнеру, что ReadyState изменилось. Для более старых модификаций контейнеров нужно вызвать Fire_ReadyStateChange(). Для современных же, Internet Explorer 4.0 и более новых, нужно обратиться к FireOnChanged().

В окне ClassView разверните CDieRoll, а затем ниже него — IDieRoll. Дважды щелкните на put_Image() и отредактируйте код этой функции. Найдите в ней приведенную ниже строку:

```
m_nReadyState = READYSTATE_LOADING;
```

Непосредственно после нее добавьте следующее:

```
Fire_ReadyStateChange();
FireOnChanged(DISPID_READYSTATE);
```

В функции put_Image () найдите такую строку:

```
m_nReadyState = READYSTATE_COMPLETE;
```

После нее добавьте те же две строки. То же самое проделайте и с функцией OnData().

Снова откомпилируйте элемент управления и вставьте его на новую страницу в Control Pad. Щелкните на игральной кости в окне Edit ActiveX Control, и после каждого щелчка будет выпадать новое число. Сохраните HTML-файл, загрузите его в Explorer и проверьте, реагирует ли кость на щелчки в то время, когда загружается файл изображения. Щелкните на Refresh, и увидите, что изображение кости обновилось, даже если вы на нем не щелкали. В качестве еще одного теста откройте контейнер ActiveX Control Test (это можно сделать с помощью меню Tools в Visual Studio) и вставьте элемент управления, а затем с помощью реестра событий (event log) убедитесь, что сообщения Click и ReadyStateChange генерируются.

Наверное, самое простое средство для тестирования подобных элементов управления — это Internet Explorer 4.0. Для того чтобы использовать его в таком качестве, нужно подключить его к Visual Studio в режиме *выполнения для отладки*. Прежде всего нужно отключить Active Desktop, если последний установлен.

Для этого сначала закройте все выполняющиеся в текущий момент приложения, поскольку придется перезагружать систему. Выберите Start⇨Settings⇨Control Panel и дважды щелкните на пиктограмме Add/Remove Programs. На вкладке Install/Uninstall выберите Microsoft Internet explorer 4.0 и щелкните на Add/Remove. Выберите последний переключатель Remove the Windows Desktop Update Component, But Keep the Internet Explorer 4.0 Web Browser. Щелкните на OK. Теперь программа Setup модифицирует системный реестр и перезагрузит систему.

После перезагрузки снова запустите Visual Studio и загрузите проект DieRollControl. Выберите Project⇨Settings и щелкните на вкладке Debug. Если по умолчанию в качестве броузера на вашем компьютере используется Internet Explorer, щелкните на стрелке рядом с Executable for Debug Session и выберите в раскрывшемся списке Default Web Browser. Если же Internet Explorer не является броузером по умолчанию, введите в поле этого списка полный путь к программе, например C:\Program Files\Internet explorer\IEXPLORE.EXE. В поле Program Arguments введите путь к HTML-файлу, который создается Control Pad для тестирования элемента управления. Щелкните на OK. Теперь каждый раз, когда вы будете выбирать команду Build⇨Start debug⇨Go или пиктограмму Go на панели инструментов, будет запускаться Explorer и в него будет загружаться страница с информацией об элементе управления. По команде Debug⇨Stop Debugging Explorer будет закрываться.

Предоставление функции DoRoll()

На следующем этапе разработки данного элемента управления необходимо экспортировать функцию, которая даст возможность контейнеру совершать броски игральной костью. Это необходимо, например, для того, чтобы менять значение, показанное на одной кости после щелчка на другой. Щелкните правой кнопкой мыши на интерфейсе IDieRoll в окне

ClassView и выберите из контекстного меню команду Add Method. Введите DoRoll в поле Method Name и оставьте раздел параметров пустым. Щелкните на OK.

Так же, как и у свойств, у функций есть dispid. Добавьте в описание enum в файле .odl константу dispidDoRoll, значение которой равно 4. Это делается для того, чтобы предотвратить конфликт с dispid функции DoRoll() в случае, если в дальнейшем вы добавите еще одно свойство. Когда вы добавляли в интерфейс функцию, после строк get и put для свойств была вставлена еще одна строка. Измените ее, чтобы использовать новый dispid:

```
[id(dispidDoRoll), helpstring("method DoRoll")] HRESULT DoRoll();
```

Текст функции DoRoll() приведен в листинге 21.20. Добавьте его в DieRoll.cpp.

Листинг 21.20. CDieRoll::DoRoll()

```
STDMETHODIMP CDieRoll::DoRoll()
{
    m_sNumber = Roll();
    FireOnChanged(dispidNumber);
    FireViewChange();
    return S_OK;
}
```

Этот код в точности похож на OnLButtonDown, но не генерирует событие Click. Вновь оттранслируйте приложение.

Протестировать этот метод можно, воспользовавшись Test Container. Откройте его с помощью команды Tools⇨ActiveX Control Test Container и выберите Edit⇨Insert New Control. Найдите Diroll Class в списке и, дважды щелкнув мышью, вставьте в контейнер игральную кость. Затем выберите Control⇨Invoke Methods. На рис. 21.14 показано диалоговое окно Invoke Methods.

Регистрация элемента управления как безопасного

В главе 20 вы добавляли элементы в реестр Registry для информирования системы о том, что новый элемент управления может безопасно принимать параметры на страницах Web и взаимодействовать со сценарием. В элементе управления ATL можно получить тот же результат, используя поддержку интерфейса IObjectSafety. Контейнер будет опрашивать этот интерфейс, чтобы узнать, безопасен ли элемент управления.

Добавьте в список наследования класса CDieRoll следующую строку:

```
public IObjectSafetyImpl<CDieRoll, INTERFACESAFE_FOR_UNTRUSTED_CALLERS |
INTERFACESAFE_FOR_UNTRUSTED_DATA>,
```

Добавьте следующую строку в карту COM:

```
COM_INTERFACE_ENTRY_IMPL(IObjectSafety)
```

После этого элемент управления автоматически станет безопасным для сценария.

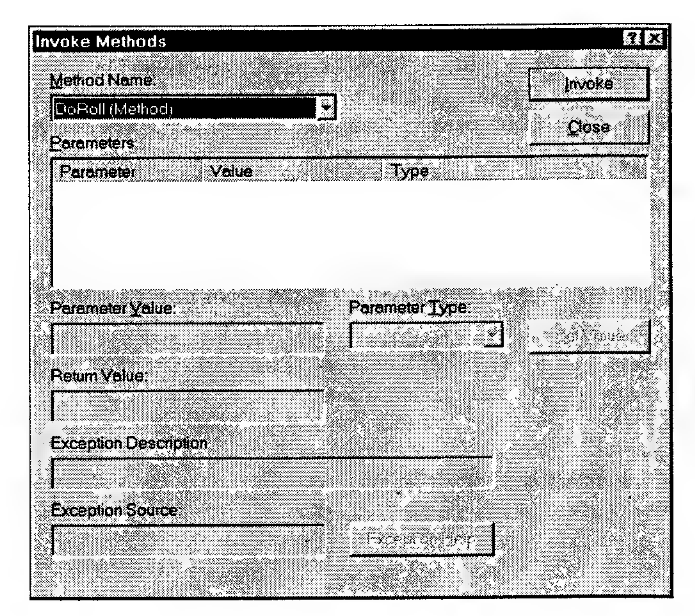


Рис. 21.14. Диалоговое окно Invoke Methods

Подготовка элемента управления к использованию в режиме разработки

Когда разработчик создает экранную форму или диалоговое окно в приложении типа Visual Basic или Visual C++, он использует палитру инструментов управления (инструментарий), в которой можно легко найти необходимые элементы управления, представленные своими пиктограммами. Поэтому следующим шагом в разработке нашего элемента управления будет создание его пиктограммы для инструментария.

Создайте ресурс растрового изображения, выбрав Insert⇨Resource и дважды щелкнув мышью на `Bitmap`. Выберите View⇨Properties и установите высоту и ширину равными 16. Измените идентификатор ресурса на `IDB_DIEROLL` и нарисуйте пиктограмму, показанную на рис. 21.15.

В реестре ссылки на этот элемент управления происходят по его номеру ресурса. Чтобы узнать, какой номер был присвоен `IDB_DIEROLL`, выберите View⇨Resource Symbols и запомните числовое значение, ассоциированное с `IDB_DIEROLL`. (На компьютере, который использовался для демонстрации примера, это число было равно 202.) Откройте `DieRoll.rgs` (файл сценария) в окне FileView и найдите такую строку:

```
ForceRemove ToolboxBitmap32 = s %MODULE%, 101
```

Измените ее следующим образом:

```
ForceRemove ToolboxBitmap32 = s %MODULE%, 202
```

Конечно, на вашем компьютере нужно использовать не 202, а полученное вами значение. Снова откомпилируйте элемент управления. Запустите `Control Pad` и выберите File⇨New⇨ HTML Layout Control. Выберите вкладку Additional инструментария и щелкните на ней правой кнопкой мыши. В появившемся контекстном меню выберите команду Additional Controls. Найдите в списке `DieRoll Class` и выберите его, затем щелкните на кнопке OK. На вкладке Additional появится новая пиктограмма (рис. 21.16).

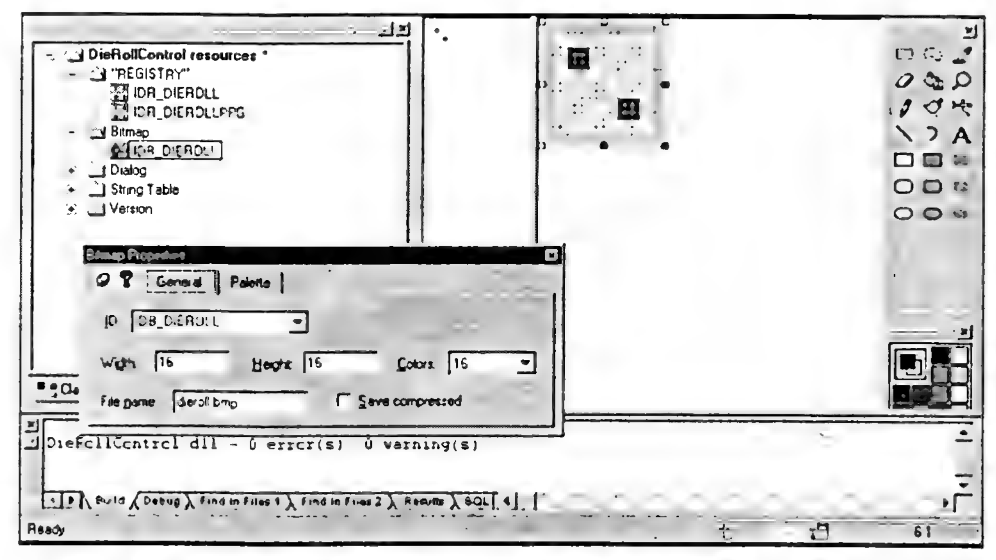


Рис. 21.15. Нарисуйте пиктограмму элемента управления

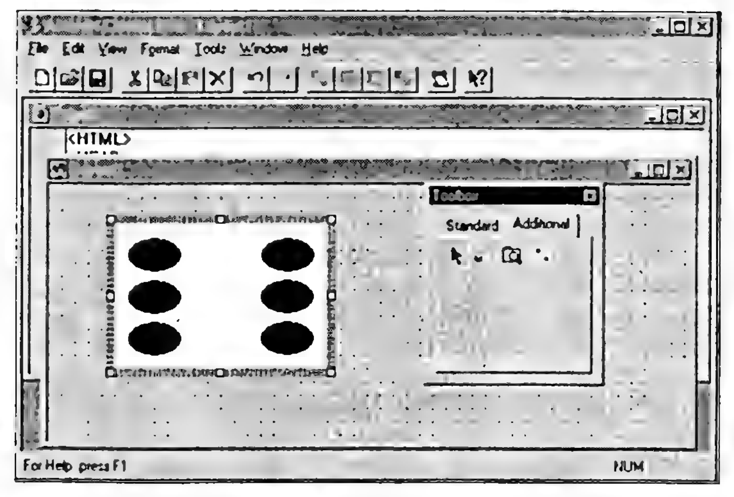


Рис. 21.16. Добавьте класс `DieRoll` на панель инструментов HTML

Минимизация размера выполняемого файла

До этого времени вы создавали отладочные версии элемента управления. Объем Dieroll.dll был равен примерно 300 Кбайт. Это намного меньше. чем 700 Кбайт файла .cab, который может потребоваться версии DieRoll на базе MFC, но это и намного больше 30 Кбайт, которые заннмает окончательная версия dieroll.ocx. После завершения разработки необходимо создать окончательную версию элемента управлення.

Выберите Build⇨Set Active Configuration, чтобы открыть диалоговое окно Set Active Project Configuration (рис. 21.17). Вы увидите, что окончательных версий в проекте ATL в два раза больше, чем в проекте MFC: кроме поддержки Unicode, необходимо также выбрать MinSize или MinDependency.

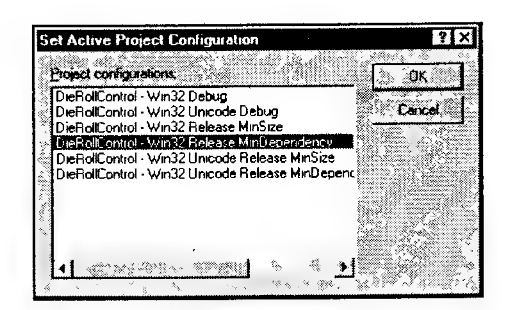


Рис. 21.17. Выбор типа компиляции в диалоговом окне Set Active Project Configuration

В распространяемой версии элемента управления минимального размера динамически компонуются библиотеки ATL DLL и ATL Registrar. В версии с минимальными зависимостями происходит статическая компоновка этих компонентов, что делает элемент управления более объемным, но полностью содержащим в себе весь код. При выборе конфигурации с минимальным размером необходимо установить файлы .cab для элемента управления и динамических библиотек так, как это обсуждалось в главе 20, в случае с библиотеками MFC. На данном этапе изучения ATL, вероятно, лучше выбрать конфигурацию с минимальными зависимостями.

Если вы выберете эту конфигурацию и начнете компоновку, то получите следующие сообщения об ошибках компоиовщика.

```
Linking...
   Creating library ReleaseMinDependency/DieRollControl.lib and object
ReleaseMinDependency/DieRollControl.exp
LIBCMT.lib(crt0.obj) : error LNK2001: unresolved external symbol _main
ReleaseMinDependency/DieRollControl.dll : fatal error LNK1120: 1 unresolved externals
Error executing link.exe.

DieRollControl.dll - 2 error(s), 0 warning(s)
```

```
Компоновка...
   Формирование библиотеки ReleaseMinDependency/DieRollControl.lib и объекта
ReleaseMinDependency/DieRollControl.exp
LIBCMT.lib(crt0.obj) : ошибка LNK2001: не найден внешний символ _main
ReleaseMinDependency/DieRollControl.dll : фатальная ошибка LNK1120: 1 внешняя ссылка не
найдена
Ошибка выполнения link.exe.

DieRollControl.dll - 2 ошибки, 0 предупреждений
```

Эти ошибки появляются не потому, что вы что-то не так сделали. По умолчанию при компиляции окончательной версии ATL используется версия tiny выполняемой библиотеки C (C runtime library — CRT), чтобы размер динамически загружаемой библиотеки был минимальным. В этой CRT нет функций `time()`, `rand()` и `srand()`, используемых при бросании кости. Компоновщик находит их в полной версии CRT, но в этой библиотеке ожидается наличие в элементе управления функции `main()`. Поскольку ее там нет, компоновка завершается неудачей.

Этот режим настраивается ключами запуска компоновщика. Выберите Project⇨Settings. В раскрывающемся списке в верхнем левом углу выберите `Win32 Release MinDependency`. Щелкните на вкладке C/C++. Щелкните на поле Preprocessor Definitions и нажмите <End>, чтобы перейти в конец поля. Удалите флаг `_ATL_MIN_CRT`, выделенный на рис. 21.18, и запятую перед ним. При последующей компиляции ошибки компоновки не возобновятся.

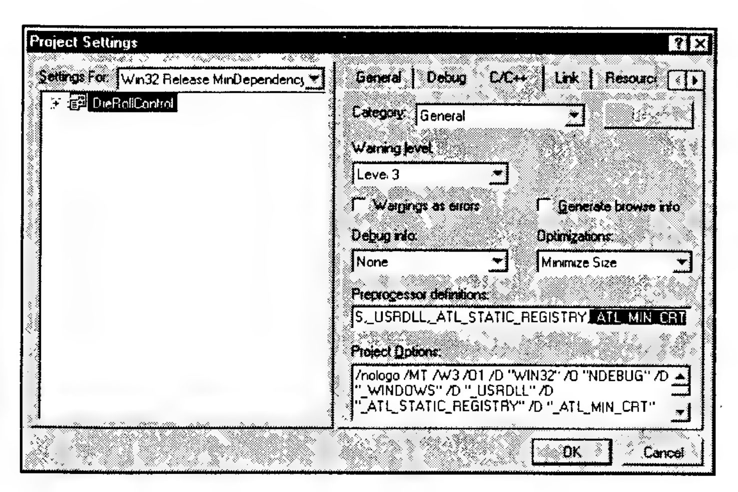


Рис. 21.18. Отключение флага, указывающего на необходимость компоновки версии tiny библиотеки C

Если так закомментировать вызовы `rand()`, `srand()` и `time()`, что элемент управления не будет работать, с флагом `_ATL_MIN_CRT`, размер модуля DLL будет равен 57 Кбайт. Если убрать флаг `_ATL_MIN_CRT`, он будет равен 86 Кбайт (значительное увеличение объема, но это существенно меньше объема версии на базе MFC и ее библиотеки). Окончательная версия, созданная при конфигурации компоновки минимального размера без флага `_ATL_MIN_CRT`, имеет размер 75 Кбайт: экономия места вряд ли стоит упаковки библиотек ATL. С закомментированными вызовами функций `rand()`, `srand()` и `time()` размер окончательной версии минимального размера с флагом `_ATL_MIN_CRT` равен всего 46 Кбайт. После удаления флага `_ATL_MIN_CRT` размер элемента управления увеличивается на 30 Кбайт. Хотя придумать другую версию этого элемента управления без использования функций `rand()`, `srand()` и `time()` вряд ли возможно, можно было бы написать свои версии этих функций и включить их в проект, чтобы элемент управления можно было компоновать с флагом `_ATL_MIN_CRT`. Алгоритмы генераторов случайных чисел и функций установки их начальных значений можно найти в книгах, посвященных алгоритмам. Функцию `time()` можно заменить функцией SDK `GetSystemTime()`. Если бы вы разрабатывали элемент управления, предназначенный для использования многими пользователями в приложениях, чувствительных ко времени, эта дополнительная работа имела бы смысл. Не забудьте, что при второй загрузке страницы Web с элементом управления ActiveX этот элемент управления не нужно загружать снова.

Использование элемента управления на странице Web

Имя и CLSID данного элемента управления отличаются от соответствующих параметров его версии на основе MFC из главы 20. Для сравнения их можно использовать одновременно на одной странице. В листинге 21.21 приведен текст в формате HTML, в котором эти два элемента управления размещены в таблице. (При создании таблицы используйте свои значения CLSID; можно воспользоваться Control Pad, как описано выше.) На рис. 21.19 показана эта страница в Explorer. Если вы хотите загрузить ее в Netscape Navigator, запустите dieroll.htm с помощью конвертера фирмы NCompass Labs, как описано в главе 20.

Листинг 21.21. Файл dieroll.htm

```
</HEAD>
<BODY>
<TABLE CELLSPACING=15>
<TR>
<TD>
Вот игральная кость MFC:<BR>
<OBJECT ID="MFCDie"
 CLASSID="CLSID:46646B43-EA16-11CF-870C-00201801DDD6"
 WIDTH="200" HEIGHT="200">
   <PARAM NAME="ForeColor" VALUE="0">
   <PARAM NAME="BackColor" VALUE="16777215">
   <PARAM NAME="Image" VALUE="beans.bmp">
Если вы видите этот текст, ваш броузер не поддерживает дескриптор OBJECT.
</OBJECT>
</TD>
<TD>
Вот игральная кость ATL:<BR>
<OBJECT ID="ATLDie" WIDTH=200 HEIGHT=200
 CLASSID="CLSID:2DE15F35-8A71-11D0-9B10-0080C81A397C">
   <PARAM NAME="Dots" VALUE="1">
   <PARAM NAME="Image" VALUE="beans.bmp">
   <PARAM NAME="ForeColor" VALUE="2147483656">
   <PARAM NAME="BackColor" VALUE="2147483653">
</OBJECT>
</TD>
</TR>
</TABLE>
</BODY>
</HTML>
```



В Visual Studio можно редактировать файлы в формате HTML так же легко, как и исходные файлы программ, используя возможность синтаксической расцветки. Просто выберите File⇨New и в списке на вкладке File выберите HTML Page. После ввода текста HTML можно щелкнуть правой кнопкой мыши в области редактирования и выбрать Preview. После этого будет запущен Explorer, а в него будет загружена HTML-страница.

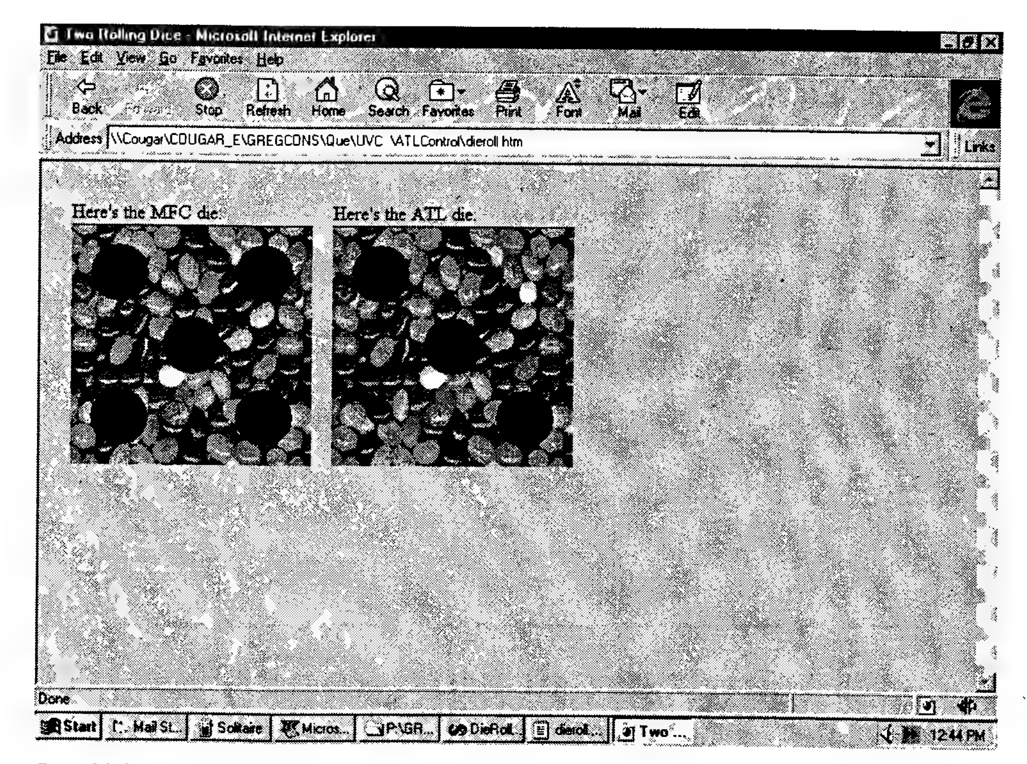


Рис. 21.19. Вместо элемента управления на основе MFC можно везде использовать элемент управления на базе ATL

Пустая страница

ЧАСТЬ VI

Современные методы программирования

В этой части...

ГЛАВА 22

Доступ к базам данных

В этой главе...

Основные понятия теории баз данных

Создание БД-программы на основе классов ODBC

Выбор между классами OBDC и DAO

OLE DB

Без сомнения, базы данных являются наиболее популярными компьютерными приложениями. Они находят применение фактически в любом виде коммерческой деятельности, начиная от создания списков покупателей и заканчивая платежными ведомостями компании. Не так давно существование множества различных систем управления базами данных (СУБД), каждая из которых определяет свои собственные структуры файлов и механизмы работы с ними, создавало огромные сложности для программистов, разрабатывающих приложения, работающие с базой данных (*БД-приложения*), поскольку необходимо было учитывать все тонкости механизма доступа к данным, хранящимся в файлах различных типов.

Современные версии Visual C++ включают классы, базирующиеся на механизмах ODBC (Open Database Connectivity — открытая связь с базами данных) и DAO (Data Access Objects — объекты доступа к данным). Верите вы или нет, но с помощью мастера AppWizard можно создать простое БД-приложение, не написав ни единой строчки текста на языке C++. Более сложные задачи все же потребуют, что называется, чистого программирования, но в значительно меньшем объеме, нежели вы себе это представляете.

Данная глава является введением в программирование с использованием ODBC-классов Visual C++. Вы также узнаете о сходстве и различиях между классами ODBC и DAO. В качестве примера будет создано БД-приложение, способное не только отображать записи, хранящиеся в базе данных, но и обновлять, добавлять, удалять, сортировать записи, а также выполнять выборку с использованием фильтров.

Основные понятия теории баз данных

Прежде чем приниматься за разработку БД-приложений, необходимо иметь представление о принципах функционирования баз данных. С момента своего изобретения СУБД прошли долгий путь, так что узнать вам предстоит многое. В этом разделе дается представление об основных концепциях теории баз данных, включая определение двух основных моделей баз данных — однофайловой и реляционной.

Однофайловая модель базы данных

В простейшем представлении *база данных* является набором записей. Каждая запись в базе данных состоит из полей, а в каждом поле содержится информация, связанная с этой конкретной записью. Например, представим себе базу данных, содержащую информацию об адресах клиентов. В такой базе данных каждому клиенту соответствует одна запись. Она состоит из шести полей: ИМЯ, УЛИЦА, ДОМ, ГОРОД, ИНДЕКС (почтовый индекс) и ТЕЛЕФОН (номер телефона). Одна из записей этой базы данных могла бы выглядеть следующим образом:

```
ИМЯ: Паниковский Михаил Самуэлевич
УЛИЦА: Прорезная
ДОМ: 1
ГОРОД: Киев
ИНДЕКС: 252001
ТЕЛЕФОН: 223-322-22
```

Вся база данных будет состоять из большого количества подобных записей, каждая из которых содержит информацию о каком-то конкретном человеке. Для того чтобы найти адрес или номер телефона конкретного клиента, в базе данных организуется поиск по полю ИМЯ. Если нужное имя будет найдено, то будет найдена и вся информация об этом клиенте, содержащаяся в записи.

Данный тип СУБД использует *однофайловую модель базы данных* (flat database model). Простая однофайловая база данных может быть достаточно мощным инструментом для до-

машнего применения или для небольшой коммерческой организации. Однако в случае больших баз данных, записи которых содержат десятки или даже сотни полей, однофайловая модель может привести к дублированию информации и нерациональному использованию памяти. Предположим, что вы управляете большим универмагом и хотите хранить некоторую информацию о служащих, включая их имена, отделы, в которых они работают, имена их менеджеров и т.д. Если в отделе "Спортивные товары" работает десять человек, то имя менеджера этого отдела будет повторяться в каждой из десяти записей. Если будет назначен новый руководитель отдела, потребуется обновить каждую из десяти записей. Все было бы гораздо проще, если бы каждая запись о служащем была связана (установлено ее *отношение*) с записью другой базы данных, содержащей информацию об отделах и их менеджерах.

Реляционная модель базы данных

Реляционная (от *relation* — *отношение, связь*) *база данных* подобна нескольким связанным однофайловым базам данных. При работе с реляционной базой данных можно не только выполнять поиск отдельных записей, как это делается в однофайловой базе данных, но и устанавливать связь одного набора записей с другим. Это позволяет организовать гораздо более эффективное хранение данных. Каждый набор записей в реляционной базе данных называется *таблицей* (table). Связи устанавливаются с использованием *ключей* (key), т.е. полей, значения которых характеризуют запись. (Например, идентификационный код служащего мог бы быть ключом для таблицы служащих.)

Реляционная база данных, используемая в данной главе в качестве примера, была создана с помощью СУБД Microsoft Access. Эта база данных является простейшей системой учета сотрудников, их руководителей и отделов, в которых они работают. На рис. 22.1–22.3 показано содержимое ее таблиц: в таблице `Employees` (Служащие) содержится информация о каждом сотруднике универмага, в таблице `Managers` (Руководители) — данные о каждом руководителе отдела универмага и в таблице `Departments` (Отделы) — сведения о самих отделах. (Эта база данных слишком проста и едва ли пригодна к использованию на практике.)

Доступ к базе данных

Доступ к данным в реляционных СУБД осуществляется с использованием определенного командного языка. Чаще всего для доступа к базам данных используется язык *SQL*, который применяется при работе не только с СУБД для настольных компьютеров, но и с громадными базами данных, используемыми в банках, учебных заведениях, промышленных корпорациях и других учреждениях, в которых необходимо решать сложнейшие задачи обработки информации. Используя язык, подобный SQL, можно извлекать данные, представленные полями записей из одной или нескольких таблиц, которые объединены в реляционную базу данных.

Однако детальное описание SQL займет довольно много места и времени и выходит за рамки тематики данной книги (кроме, разве что, этой главы). Фактически целые курсы в колледжах посвящаются изучению методов разработки и реализации баз данных, а также управления ими. Поскольку данная глава слишком мала для изучения методов работы с реляционной базой данных на каком-нибудь полезном примере, мы используем таблицу `Employees` (см. рис. 22.1) базы данных универмага для разработки простой БД-программы. Закончив работу над этим приложением, вы освоите один из методов обновления таблицы реляционной базы данных, который не предполагает использования команд на языке SQL. (Те из вас, кто жить не могут без SQL, получат истинное наслаждение от чтения главы 23.)

Microsoft Access

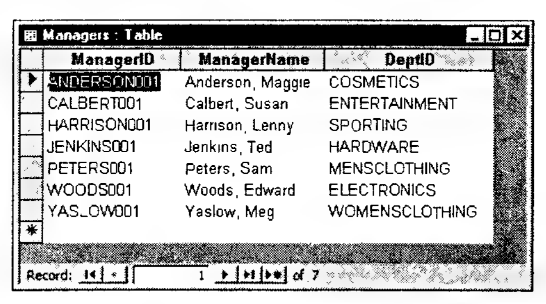
Employees : Table

EmployeeID	EmployeeName	EmployeeRate	DeptID
ANDERSON00	Anderson, Richard	6.53	MENSCLOTHING
GREENE001	Greene, Nancy	6.55	SPORTING
HANLEY001	Hanley, Frank	7.25	HARDWARE
JACKSON001	Jackson, Ken	5.75	MENSCLOTHING
JOHNSON001	Johnson, Ed	7.10	HARDWARE
JOHNSON002	Johnson, Mary	7.35	COSMETICS
KELLY001	Kelly, Mick	7.10	HARDWARE
LITTLETON001	Littleton, Sarah	6.10	WOMENSCLOTHING
NEBBICK001	Nebbick, Lucy	5.90	ENTERTAINMENT
OLSEN001	Olsen, Jane	6.88	WOMENSCLOTHING
PERRY001	Perry, Cal	5.36	ENTERTAINMENT
SANFORD001	Sanford, Faith	6.55	ELECTRONICS
SMITH001	Smith, James	5.75	ELECTRONICS
ULEY001	Uley, Victor	5.26	SPORTING
WHITE001	White, Gail	6.22	COSMETICS
WILSON001	Wilson, Denny	6.20	SPORTING

Record: 1 of 16

Datasheet View

Рис. 22.1. Таблица `Employees` *содержит поля данных о каждом сотруднике универмага*



Managers : Table

ManagerID	ManagerName	DeptID
ANDERSON001	Anderson, Maggie	COSMETICS
CALBERT001	Calbert, Susan	ENTERTAINMENT
HARRISON001	Harrison, Lenny	SPORTING
JENKINS001	Jenkins, Ted	HARDWARE
PETERS001	Peters, Sam	MENSCLOTHING
WOODS001	Woods, Edward	ELECTRONICS
YASLOW001	Yaslow, Meg	WOMENSCLOTHING

Record: 1 of 7

Рис. 22.2. Таблица `Managers` *содержит информацию о каждом руководителе отдела универмага*

Departments : Table

DeptID	DeptName	ManagerID
COSMETICS	Cosmetics	ANDERSON001
ELECTRONICS	Electronics	WOODS001
ENTERTAINMENT	Entertainment	CALBERT001
HARDWARE	Hardware	JENKINS001
MENSCLOTHING	Men's Clothing	PETERS001
SPORTING	Sporting Goods	HARRISON001
WOMENSCLOTHING	Women's Clothing	YASLOW001

Record: 1 of 7

Рис. 22.3. Таблица `Departments` *содержит данные о каждом отделе универмага*

Классы ODBC Visual C++

Создавая с помощью мастера Visual C++ AppWizard программу, работающую с базами данных, вы получаете в итоге приложение, широко использующее различные классы ODBC из состава библиотеки MFC. Наиболее важными из этих классов являются CDatabase, CRecordset и CRecordView.

Мастер AppWizard автоматически генерирует текст программы, необходимый для создания объекта класса CDatabase. Этот объект обеспечивает связь между создаваемым приложением и источником данных, с которым оно работает. В большинстве случаев использование класса CDatabase в программах, сгенерированных AppWizard, прозрачно для программиста. Вся необходимая обработка обеспечивается самой системой управления.

Кроме того, AppWizard генерирует текст программы создания объекта класса CRecordset, используемого в приложении. Объект класса CRecordset представляет собой реальные данные, выбранные в настоящий момент из источника данных, а его методы обеспечивают выполнение операций с этими данными. В дальнейшем данные, выбранные программой в текущий момент, будем называть *выборкой данных*.

И наконец, объект класса CRecordView в БД-программе занимает место объекта класса представления, с которым постоянно приходится иметь дело в приложениях, созданных с помощью мастера AppWizard. Окно, создаваемое объектом класса CRecordView, подобно диалоговому окну, выполняющему роль средства общения пользователя с приложением. Это диалоговое окно обеспечивает в приложении связь с объектом класса CRecordset, осуществляя обмен информацией между программой, элементами управления окна и выборкой данных. Когда с помощью мастера AppWizard создается новое БД-приложение, на программиста возлагается обязанность поместить в окно объекта CRecordView элементы управления, способные выполнять ввод и редактирование данных. Как правило, это текстовые поля. Такие элементы управления следует связать с полями записей базы данных, которые они представляют, для того, чтобы приложение направляло данные, выбранные для просмотра, куда следует.

В следующем разделе мы рассмотрим шаг за шагом процесс разработки приложения Employee, и вы увидите, как взаимодействуют между собой объекты различных классов, о которых шла речь выше.

Создание БД-программы на основе классов ODBC

Создать с помощью Visual C++ простую БД-программу, использующую классы ODBC, совсем несложно — для этого нужно выполнить всего несколько операций.

1. Зарегистрировать базу данных в системе.
2. Используя мастер AppWizard, создать заготовку БД-приложения.
3. Добавить в заготовку приложения программный код, реализующий функции, которые AppWizard автоматически не формирует.

В следующих разделах вы узнаете, как это делается практически на примере создания приложения Employee, предназначенного для просмотра, добавления, удаления, обновления и сортировки записей таблицы Employees, входящей в состав нашей экспериментальной базы данных для универмага.

Регистрация базы данных

Прежде чем приступить к созданию БД-приложения, необходимо базу данных, с которой вы хотите работать, зарегистрировать в качестве источника данных, доступ к которому будет осуществляться через драйвер ODBC. Для этого выполните следующие действия.

1. Создайте на жестком диске папку с именем `Database` и скопируйте в нее с Web-страницы файл `DeptStore.mdb`. Если у вас нет доступа к сети Internet, можете самостоятельно сформировать в Microsoft Access три таблицы.

   Файл `DeptStore.mdb` представляет собой базу данных, созданную в СУБД Microsoft Access. Эта база данных будет использоваться для приложения `Employee` в качестве источника данных.

2. Из меню Start в Windows 95 откройте Control Panel. В ее окне сделайте двойной щелчок на пиктограмме 32-Bit ODBC. Раскроется диалоговое окно ODBS Data Source Administrator, показанное на рис. 22.4. Если в представленном в этом окне списке отсутствует необходимый вам драйвер (в данном примере — Microsoft Access Driver), выполните следующий пункт. В противном случае перейдите сразу к п. 4.

3. Щелкните на кнопке Add (Добавить). Появится диалоговое окно Create New Data Source (Создать новый источник данных). Из списка драйверов выберите `Microsoft Access Driver`, как показано на рис. 22.5, а затем щелкните на кнопке Finish (Конец).

   Теперь Microsoft Access Driver является тем ODBC-драйвером, который будет связан с источником данных, предназначенным для работы с приложением `Employee`.

4. В диалоговом окне ODBS Data Source Administrator выделите строку `Microsoft Access 97 Database` и щелкните на кнопке Configure. Когда раскроется диалоговое окно ODBC Microsoft Access 97 Setup, введите значение `Department Store` в поле Data Source Name (Имя источника данных) и значение `Department Store Sample` в поле Description (Описание), как показано на рис. 22.6.

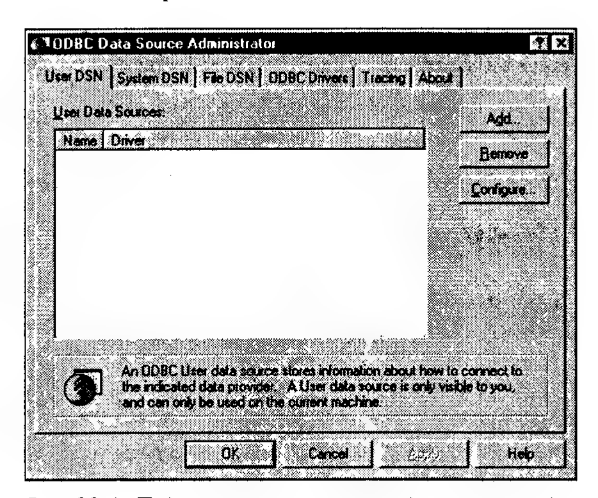


Рис. 22.4. Подключение источника данных к создаваемому приложению начинается с вызова на экран окна ODBC Data Source Administrator

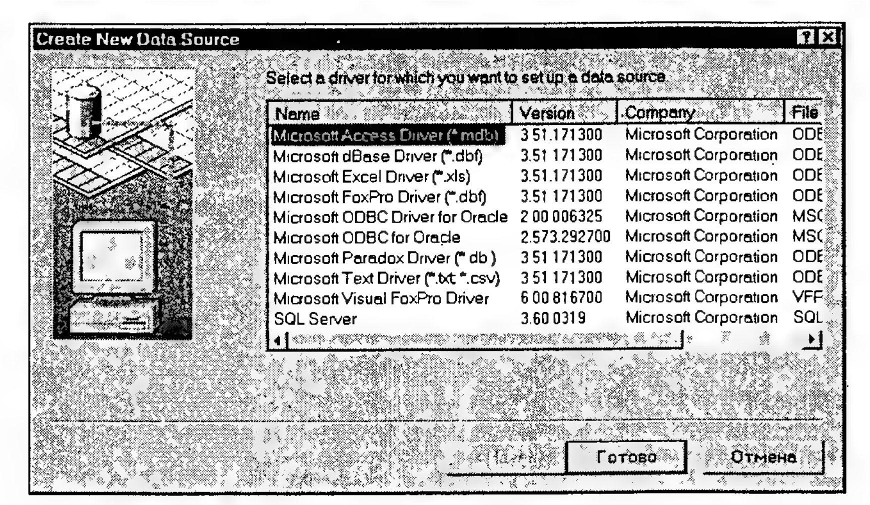


Рис. 22.5. Создание нового источника данных сводится к выбору драйвера `Microsoft Access Driver` *из имеющегося списка*

Имя источника данных предназначено для идентификации каждого из создаваемых источников данных. Поле Description дает возможность ввести более детальную информацию о созданном источнике данных.

5. Щелкните на кнопке Select (Выбрать). Появится окно Select Database (Выбор базы данных), предназначенное для поиска и выборки файла. Отыщите на жестком диске и выберите файл `DeptStore.mdb` (рис. 22.7).

6. Для завершения работы по выбору базы данных щелкните на кнопке OK, а затем для завершения процесса создания источника данных щелкните на кнопке OK в диалоговом окне ODBC Microsoft Access 97 Setup. И наконец, щелкните на кнопке OK в диалоговом окне ODBC Data Source Administrator.

ODBC Microsoft Access 97 Setup
Data Source Name: Department Store
Description: Department Store Sample
Database
Database:
Select... Create... Repair... Compact...
System Database
None
Database:
OK
Cancel
Help
Advanced...
Options>>

Рис. 22.6. Назовите источник данных по собственному желанию

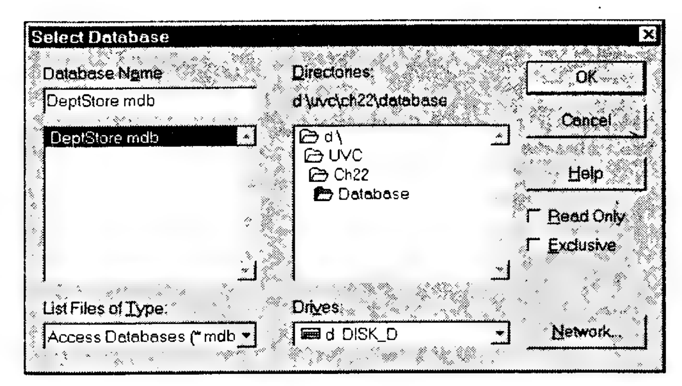


Рис. 22.7. Выполните поиск файла с расширением `.mdb`*, в котором находятся данные для приложения*

Теперь в системе установлен доступ к файлу базы данных `DeptStore.mdb` с помощью драйвера ODBC `Microsoft Access Driver`.

Создание заготовки для приложения `Employee`

Теперь, когда источник данных создан и зарегистрирован, пришло время создания заготовки приложения `Employee`. Эта процедура состоит из нескольких этапов, которые будут описаны ниже.

1. В строке меню Visual Studio выберите команду File⇨New. Щелкните на корешке вкладки Projects.
2. В списке выберите значение `MFC AppWizard (exe)` и введите значение `Employee` в поле Name, как показано на рис. 22.8. Щелкните на OK. Появится диалоговое окно MFC AppWizard Step 1.

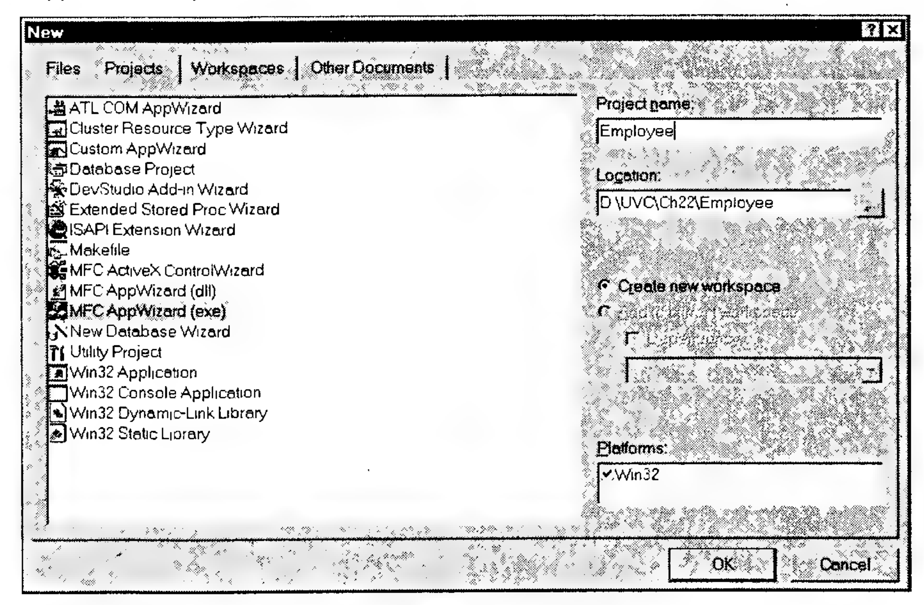


Рис. 22.8. Создайте с помощью мастера `AppWizard` *обыкновенное приложение на основе библиотеки классов MFC*

3. Для того чтобы гарантировать, что приложение Employee не позволит открыть более одного окна, установите флажок опции Single document, как показано на рис. 22.9. Щелкните на кнопке Next.

4. Для того чтобы AppWizard сгенерировал классы, необходимые для просмотра содержимого базы данных, установите переключатель Database view without file support (Просмотр базы данных без поддержки работы с файлами), как показано на рис. 22.10. Приложение не будет создавать или использовать никаких дополнительных файлов, кроме файла базы данных, поэтому оно не нуждается в поддержке работы с файлами (механизма сохранения-восстановления). Для подсоединения приложения к созданному ранее источнику данных щелкните на кнопке Data Source (Источник данных).

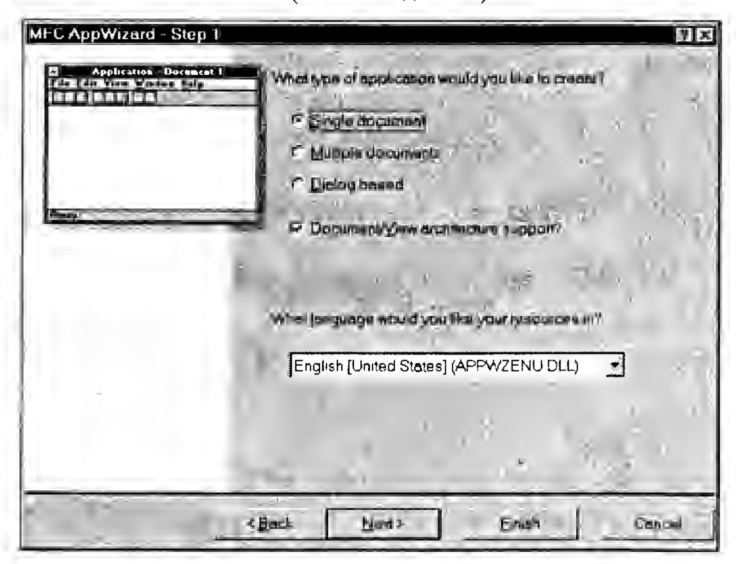


Рис. 22.9. Создаваемое приложение будет иметь только одно окно

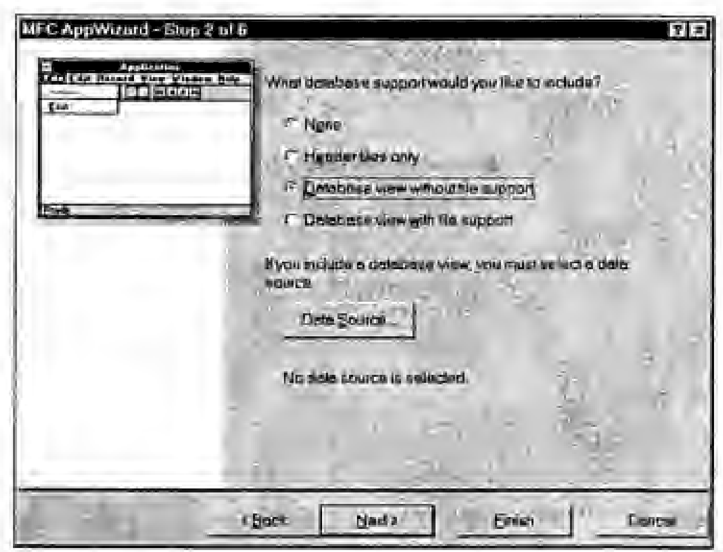


Рис. 22.10. Настройка для просмотра базы данных без поддержки работы с другими типами файлов

5. Разверните в диалоговом окне Database Options (Опции базы данных) список источников данных ODBC и выберите источник данных Department Store, как показано на рис. 22.11. Щелкните на кнопке OK.

6. Выберите в диалоговом окне Select Database Tables (Выбор таблиц базы данных) таблицу Employees (рис. 22.12) и щелкните на OK. Вновь раскроется диалоговое окно Step 2, в котором сразу под кнопкой Data Source будут выведена информация о подключенном источнике данных, как показано на рис. 22.13.

   Теперь таблица Employees источника данных Department Store связана с создаваемым приложением Employee. Щелкните на кнопке Next и переходите к следующему этапу настройки.

7. Щелкнув на кнопке Next, примите установку по умолчанию No compound document support (Не поддерживать составные документы).

8. В диалоговом окне MFC AppWizard - Step 4 of 6 сбросьте флажок опции Printing and print preview (Печать и предварительный просмотр распечатки), после чего диалоговое окно должно выглядеть так, как показано на рис. 22.14. Щелкните на кнопке Next.

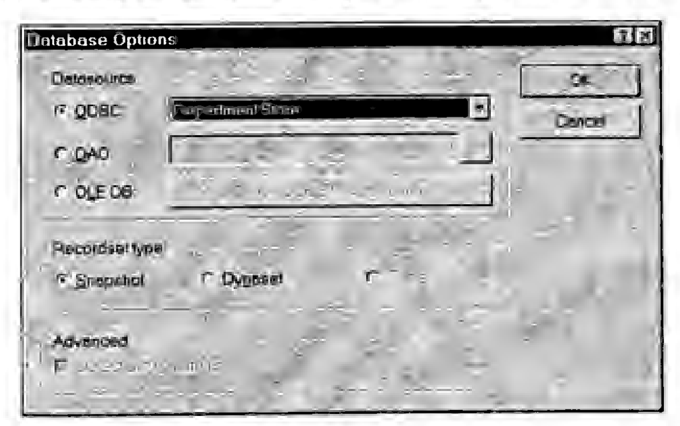


Рис. 22.11. Выбор для приложения источника данных Department Store

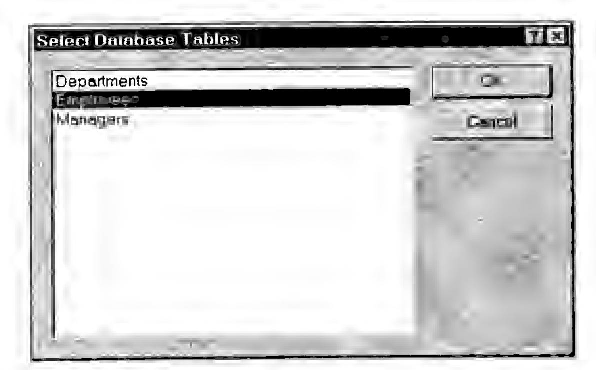


Рис. 22.12. Выбор из источника данных таблицы, которая будет использоваться в создаваемом приложении

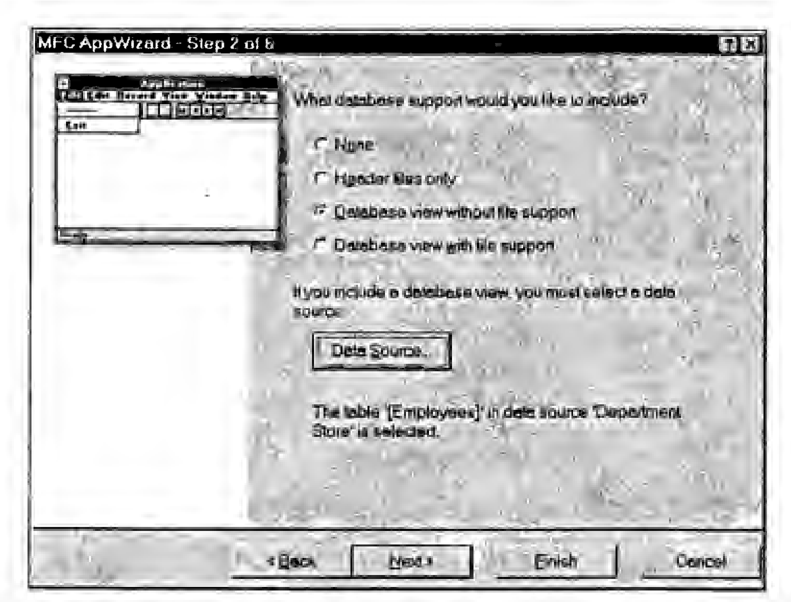


Рис. 22.13. Диалоговое окно второго этапа после выбора источника данных

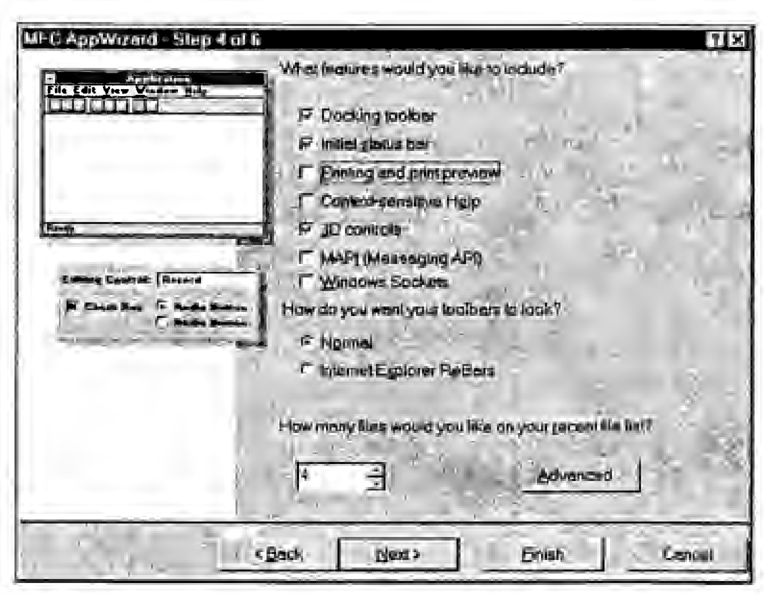


Рис. 22.14. Отключение поддержки печати

9. Щелкнув на кнопке Next, примите установки по умолчанию в окне MFC AppWizard - Step 5 of 6. На следующем, шестом этапе, просто щелкните на кнопке Finish, завершив процесс задания установок для приложения `Employee`. Раскрывшееся диалоговое окно New Project Information должно выглядеть так, как показано на рис. 22.15.

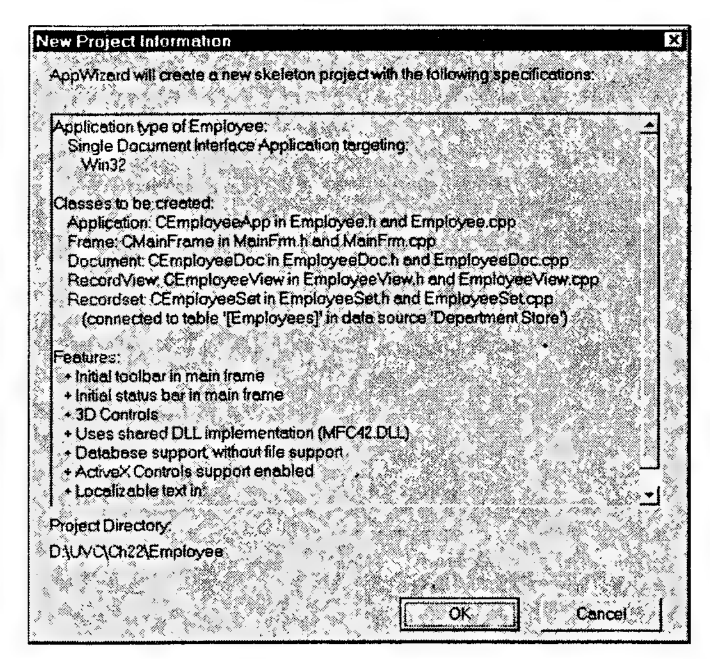


Рис. 22.15. В сводке установок, выполненных для приложения, наряду с обычной информацией упоминается источник данных

10. Щелкните на кнопке OK, после чего мастер AppWizard создаст заготовку приложения Employee.

На данный момент уже можно откомпилировать создаваемое приложение, либо щелкнув на пиктограмме Build панели инструментов, либо выбрав в меню команду Build⇨Build, либо нажав на клавиатуре клавишу <F7>. После завершения компиляции можно запустить программу на выполнение, либо выбрав на панели меню команду Build⇨Execute, либо нажав клавиши <Ctrl+F5>. Когда программа будет запущена, раскроется ее окно, показанное на рис. 22.16. Обратите внимание на то, что на панели инструментов заготовки приложения уже имеются органы управления навигацией по записям таблицы, однако в окне приложения никаких данных нет, поскольку не созданы соответствующие элементы управления и не выполнено их связывание с полями таблицы Employees, которые требуется просматривать. Как это сделать, будет описано в следующем разделе.

Создание экранной формы для отображения содержимого базы данных

Следующим шагом в разработке БД-приложения Employee будет модификация его экранной формы, предназначенной для отображения данных в окне приложения. Поскольку эта форма является просто специализированным типом диалогового окна, модификацию можно легко осуществить с помощью редактора ресурсов Visual Studio, в чем вы убедитесь, выполнив следующие операции.

1. Для отображения ресурсов приложения щелкните на корешке вкладки ResourceView.
2. Разверните дерево ресурсов, щелкнув на знаке "+" перед папкой Employee Resources. Далее аналогичным образом откройте папку ресурсов Dialog. Сделайте двойной щелчок на иден-

тификаторе диалогового окна IDD_EMPLOYEE_FORM и тем самым откройте диалоговое окно в редакторе ресурсов, как показано на рис. 22.17.

3. Выделите, щелкнув на ней, строку в центре диалогового окна, а затем удалите ее, нажав клавишу <Del>.

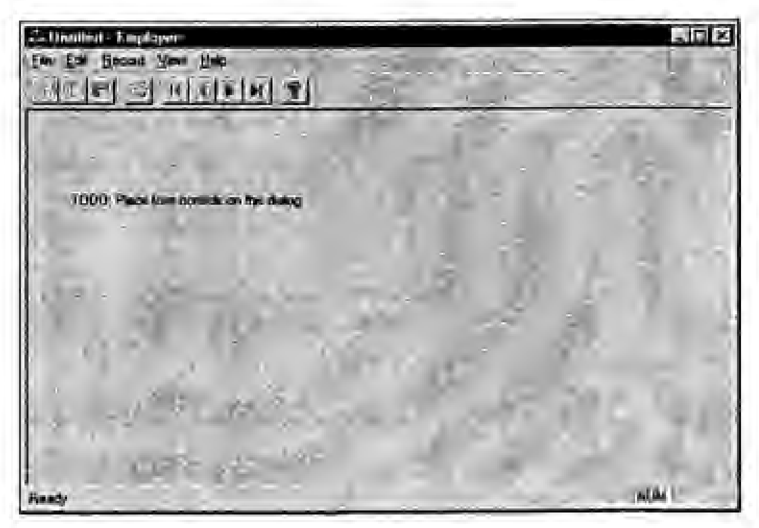


Рис. 22.16. Заготовка приложения Employee *выглядит неплохо, но практически ничего не выполняет*

Рис. 22.17. Диалоговое окно, открытое в редакторе ресурсов

4. Пользуясь инструментами редактора диалогового окна, добавьте в него текстовые поля редактирования и статические надписи по образцу, показанному на рис. 22.18. (Редактирование диалоговых окон описано в главе 2.) Присвойте полям редактирования следующие идентификаторы: IDC_EMPLOYEE_ID, IDC_EMPLOYEE_NAME, IDC_EMPLOYEE_RATE, IDC_EMPLOYEE_DEPT. Для текстового поля IDC_EMPLOYEE_ID установите стиль Read-Only (определяется одноименным флажком на вкладке Styles в окне свойств Edit Properties).

5. Каждое из этих текстовых полей будет представлять поле записи базы данных. Атрибут Read-Only (только для чтения) установлен для первого (текстового) поля по той причине, что оно будет содержать первичный ключ базы данных, который не подлежит изменению.

6. Для вызова мастера ClassWizard выберите команду View⇨ClassWizard и в раскрывшемся окне щелкните на корешке вкладки Member Variables.

7. Выбрав ресурс IDC_EMPLOYEE_DEPT, щелкните на кнопке Add Variable. Раскроется диалоговое окно Add Member Variable.

8. Щелкните на стрелке рядом с раскрывающимся списком Member Variable Name и выберите в нем значение m_pSet->m_DeptID, как показано на рис. 22.19.

9. Аналогично свяжите с элементами редактирования остальные переменные-члены (m_pSet->EmployeeID, m_pSet->EmployeeName и m_pSet->EmployeeRate). Когда это будет сделано, вкладка Member Variables окна MFC ClassWizard должна выглядеть так, как показано на рис. 22.20.

   Выбрав переменные-члены класса приложения CEmployeeSet (производного от класса MFC CRecordset) в качестве переменных для элементов управления в классе представления базы данных (в форме), вы установили связь, посредством которой может происходить обмен данными между элементами редактирования и источником данных.

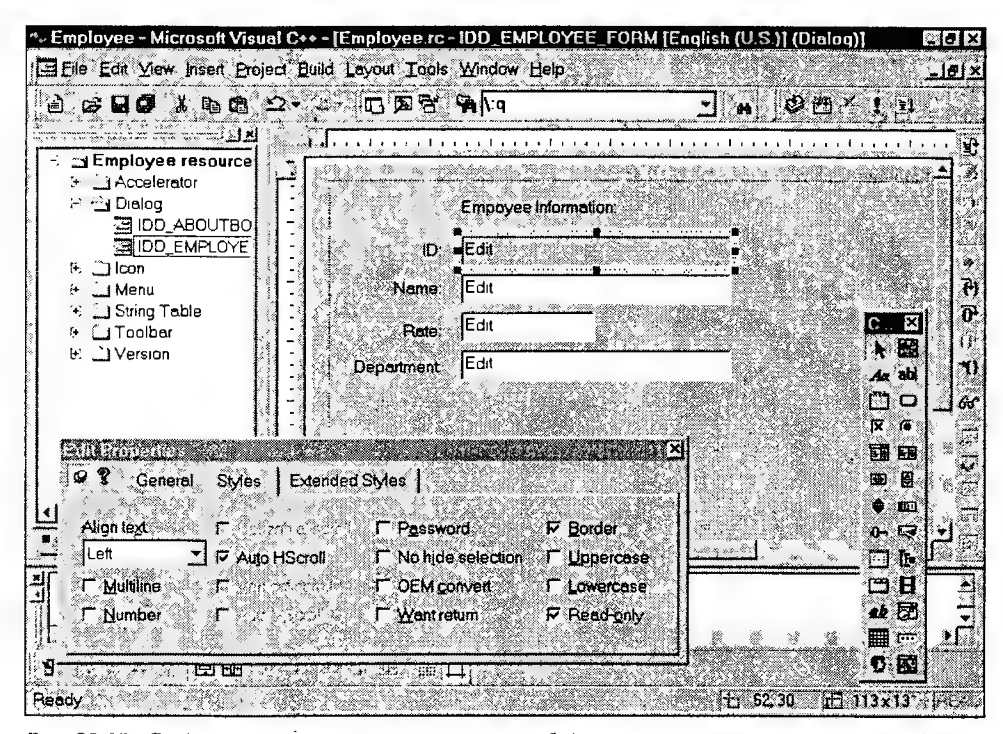


Рис. 22.18. Создание диалогового окна, которое будет использоваться в качестве формы для базы данных

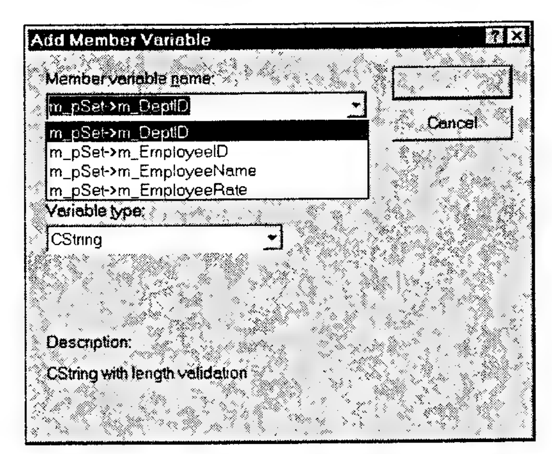


Рис. 22.19. Связывание поля `IDC_EMPLOYEE_DEPT` *с переменной-членом* `m_DeptID` *класса выборки данных*

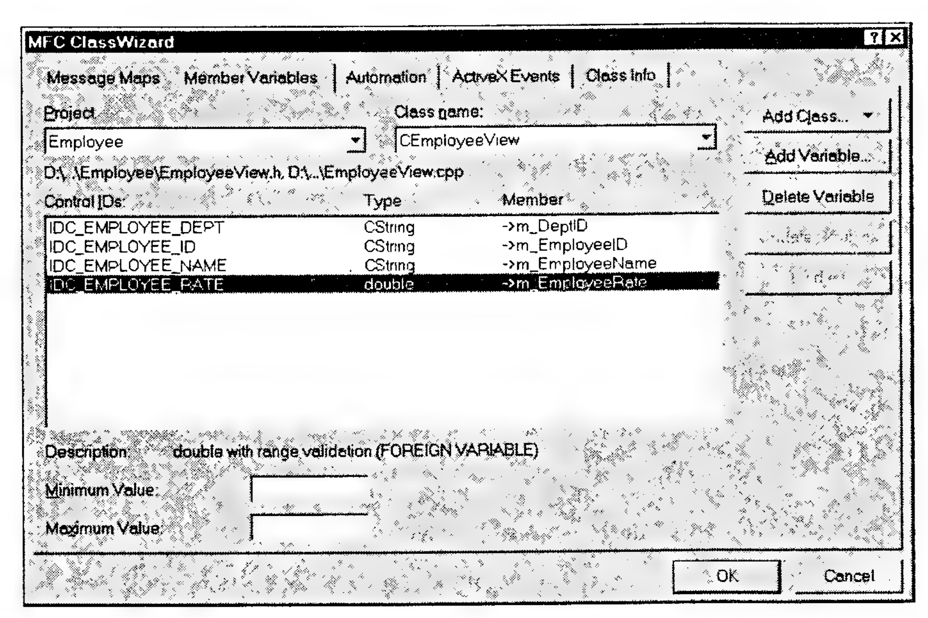


Рис. 22.20. Все четыре элемента управления связаны с переменными-членами класса `CEmployeeSet`

10. После щелчка на кнопке OK в окне MFC ClassWizard внесенные изменения будут зафиксированы в тексте программы.

Мы завершили создание экранной формы для отображения данных в приложении `Employee`. Оттранслируйте и запустите программу еще раз, и вы увидите окно, показанное на рис. 22.21. Теперь наше приложение отображает содержимое записей таблицы `Employee`. Используя элементы управления навигацией, расположенные на панели инструментов приложения, можно перемещаться от одной записи таблицы `Employee` к другой.

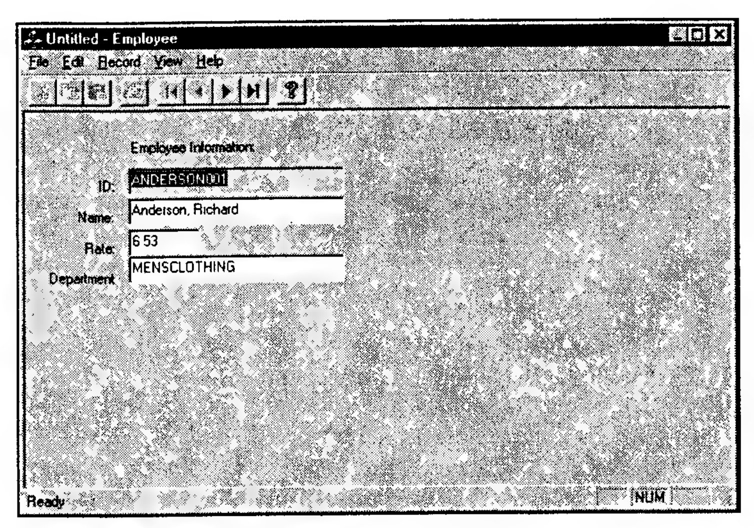


Рис. 22.21. Теперь приложение отображает в своем окне данные из таблицы Employee

Проверив возможность перемещения в базе данных, попробуйте обновить любую из записей. Для этого достаточно просто изменить содержимое любого из полей записи (за исключением поля EmployeeID, которое является первичным ключом и не может быть изменено). При переходе к другой записи приложение автоматически перенесет отредактированные данные в таблицу. Команды меню Record (Запись) приложения позволяют перемещаться по записям в базе данных точно так, как пиктограммы панели инструментов.

Обратите внимание на то, что достаточно сложная программа доступа к базе данных была создана без ввода хотя бы одной строчки текста программы на C++, а это вызывает приятное удивление. Однако возможности приложения Employee достаточно ограничены. Оно, например, не позволяет добавлять или удалять записи. Включение этих функций в создаваемое приложение будет описано в следующем разделе.

Добавление и удаление записей

Когда мы включим в создаваемое приложение возможность добавлять и удалять записи в таблице базы данных, оно превратится в полнофункциональную программу обработки однофайловой (но не реляционной) базы данных. В нашем случае в роли однофайловой базы данных выступает таблица Employee реляционной базы данных универмага. Добавление и удаление записей в таблице базы данных реализуется достаточно просто благодаря существованию в Visual C++ классов CRecordView и CRecordset, предоставляющих все необходимые методы для выполнения этих стандартных операций. Необходимо будет добавить в приложение несколько команд меню. О том, как это делается, уже рассказывалось в главе 8. Для добавления к приложению Employee команд Add (Добавить) и Delete (Удалить) выполните следующие действия.

1. Щелкните на корешке вкладки ResourceView, откройте папку Menu и сделайте двойной щелчок на меню IDR_MAINFRAME. На экране раскроется окно редактора меню, показанное на рис. 22.22.
2. Щелкните в меню Record и тем самым откройте его, а затем щелкните на пустой области в нижней части этого меню. Выберите команду View⇨Properties и переместите раскрывшееся диалоговое окно Properties на подходящее для него место.
3. В поле ID введите значение ID_RECORD_ADD, а в поле Caption введите значение &Add Record. В результате в меню Record будет добавлена новая команда.
4. В следующий пустой элемент меню внесите команду удаления, имеющую идентификатор ID_RECORD_DELETE (поле ID) и заголовок &Delete Record (поле Caption), как показано на рис. 22.23.

Рис. 22.22. Окно редактора меню Visual Studio располагается в правой части экрана

Рис. 22.23. Добавление в меню команд добавления и удаления записей

Далее необходимо добавить на панель инструментов пару новых пиктограмм и связать с ними эти команды (как это сделать, подробно описано в главе 9). Выполните следующие действия.

1. В дереве ресурсов в окне ResourceView откройте папку Toolbar и сделайте двойной щелчок на идентификаторе IDR_MAINFRAME. Панель инструментов приложения будет отображена в окне редактора ресурсов.

2. Щелкнув на пустой пиктограмме панели инструментов, выберите ее, а затем с помощью инструментов графического редактора нарисуйте на ней голубой знак "плюс".

3. Сделайте двойной щелчок на новой пиктограмме панели инструментов. Раскроется окно свойств Toolbar Button Properties. В списке ID выберите значение ID_RECORD_ADD, как показано на рис. 22.24.

4. Снова выделите пустую пиктограмму панели инструментов и нарисуйте на ней красный знак "минус"; присвойте пиктограмме идентификатор ID_RECORD_DELETE, как показано на рис. 22.25. Перетащите пиктограммы добавления и удаления левее пиктограммы справки, помеченной знаком вопроса.

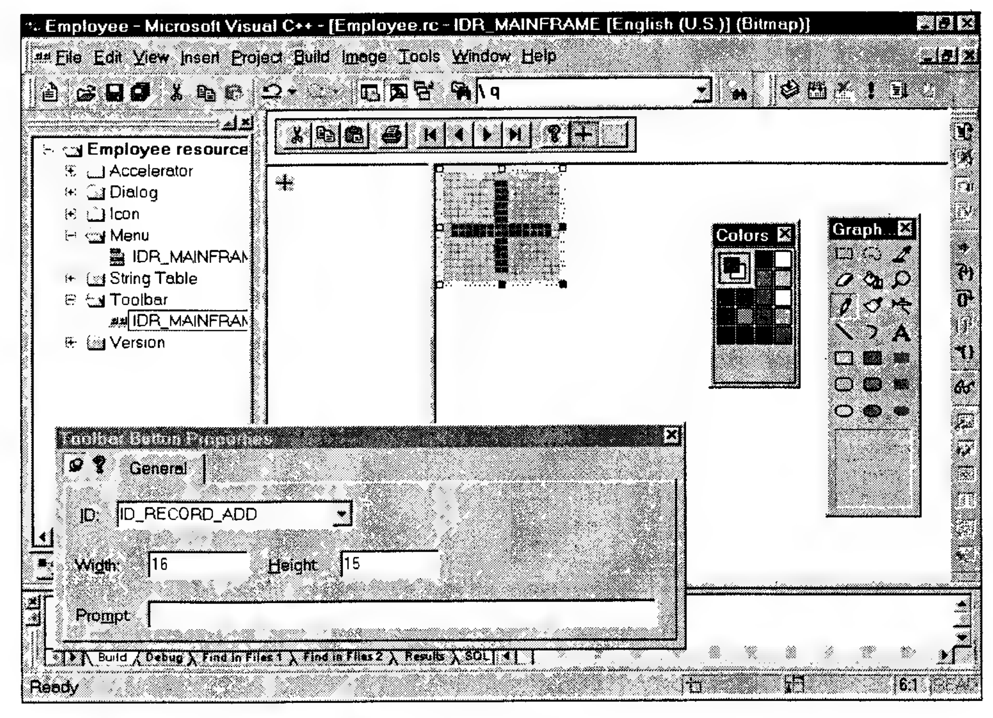


Рис. 22.24. Добавление пиктограммы и соединение ее с командой меню

Теперь, когда в меню уже добавлены новые команды и на панель инструментов помещены соответствующие пиктограммы, необходимо сформировать программный код, который будет перехватывать командные сообщения, посылаемые, когда пользователь щелкает на пиктограмме или выбирает пункт меню. Основные сведения о методике организации перехвата сообщений уже были изложены в главах 3, 8 и 9. Так как в нашем приложении с базой данных связан класс представления, в нем и следует организовать перехват этих сообщений. Выполните следующие операции.

1. Раскройте окно ClassWizard и выберите в нем вкладку Message Maps.

2. В списке Class Name выберите значение CEmployeeView, а в списке Object IDs выберите значение ID_RECORD_ADD, после чего сделайте двойной щелчок на значении COMMAND в списке Messages. Раскроется диалоговое окно Add Member Function, показанное на рис. 22.26.

Рис. 22.25. Пиктограмма со знаком "минус" будет отвечать за функцию удаления записи

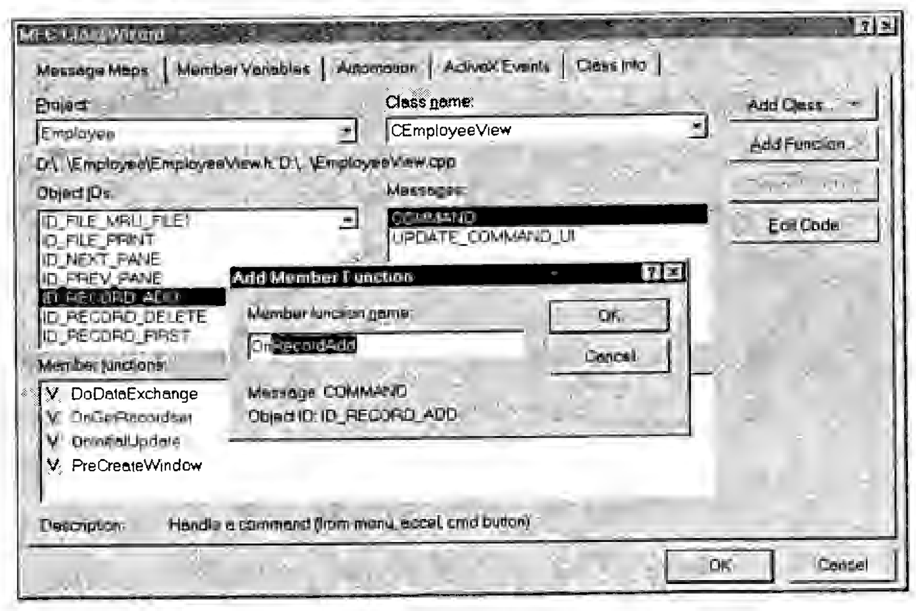


Рис. 22.26. Добавление функции перехвата сообщения

3. Щелкните на кнопке OK, приняв для новой функции имя, предложенное по умолчанию. Имя новой функции появится в списке Member Functions в нижней части окна ClassWizard.

4. Аналогичным образом добавьте метод для обработки команды ID_RECORD_DELETE. После этого список функций должен выглядеть так, как показано на рис. 22.27. Закройте окно мастера ClassWizard, щелкнув на кнопке OK.

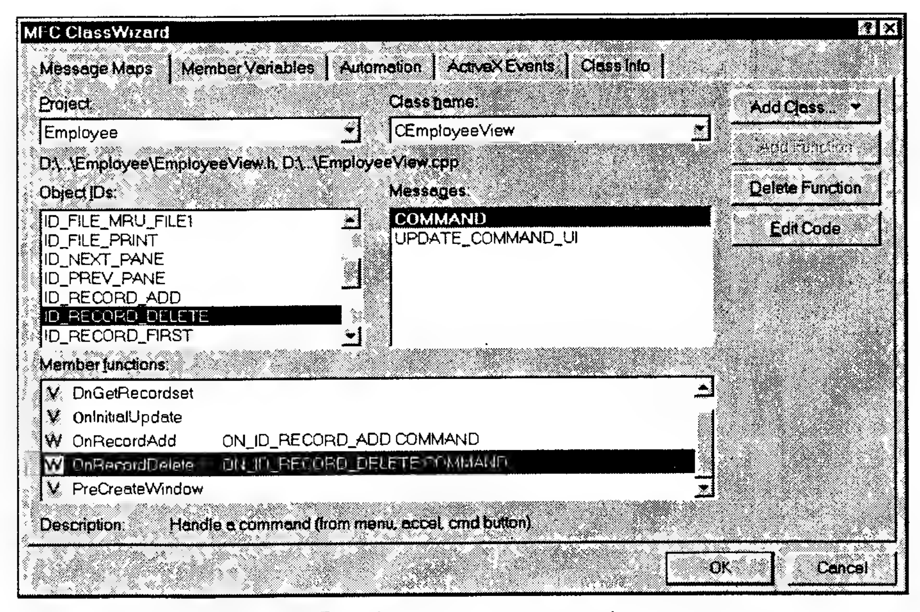


Рис. 22.27. В окне Memeber Functions *появились новые функции-члены*

5. В окне ClassView, дважды щелкнув на элементе CEmployeeView, откройте файл Employee-View.h. В объявлении класса добавьте следующие строки в раздел Attributes:

```
protected:
        BOOL m_bAdding;
```

6. В окне ClassView сделайте двойной щелчок на конструкторе класса CEmployeeView и добавьте следующую строку в конец этой функции:

```
m_bAdding = FALSE;
```

7. Сделайте двойной щелчок на функции OnRecordAdd() и отредактируйте ее текст так, как показано в листинге 22.1. Объяснения к этому коду будут даны в следующем разделе.

Листинг 22.1. Функция CEmployeeView::OnRecordAdd()

```
void CEemployeeView::OnRecordAdd()
{
        m_pSet->AddNew();
        m_bAdding = TRUE;
        CEdit* PCtrl = (CEdit*)GetDlgItem(IDC_EMPLOYEE_ID);
        int result = pCtrl->SetReadOnly(FALSE);
        UpdateData(FALSE);
}
```

8. В окне ClassView щелкните правой кнопкой мыши на элементе CEmployeeView и выберите в раскрывшемся контекстном меню команду Add Virtual Function. В левом списке выберите значение OnMove, как показано на рис. 22.28, а затем щелкните на кнопке Add and Edit. В результате в класс будет добавлена функция и можно будет немедленно отредактировать заготовку ее текста.

9. Отредактируйте функцию OnMove() так, чтобы она содержала текст программы, приведенный в листинге 22.2. Объяснения к этому коду будут даны в следующем разделе.

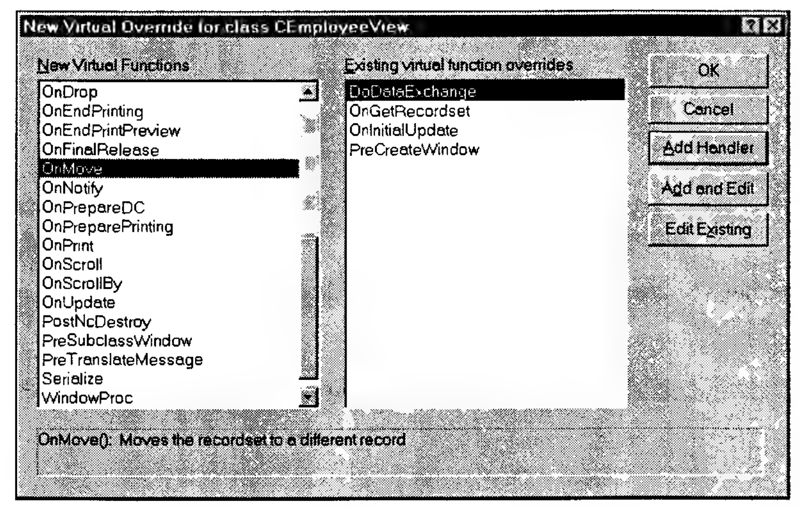


Рис. 22.28. Переопределение функции OnMove()

Листинг 22.2. Функция CEmployeeView::OnMove()

```
BOOL CEmployeeView::OnMove(UNIT nIDMoveCommand)
{
        if (m_bAdding)
        {
                m_bAdding = FALSE;
                UpdateData(TRUE);
                if (m_pSet->CanUpdate())
                        m_pSet->Update();
                m_pSet->Requery();
                UpdateData(FALSE);
                CEdit* pCtrl = (CEdit*)GetDlgItem(IDC_EMPLOYEE_ID);
                pCtrl->SetReadOnly(TRUE);
                return TRUE;
        }
        else
                return CRecordView::OnMove(nIDMoveCommand);
}
```

10. Сделайте двойной щелчок на функции OnDelete() и отредактируйте ее так, чтобы ее текст соответствовал листингу 22.3. Пояснения к этому тексту будут даны в следующем разделе.

Листинг 22.3. Функция CEmployeeView::OnRecordDelete()

```
void CEmployeeView::OnRecordDelete()
{
        m_pSet->Delete();
        m_pSet->MoveNext();

        if (m_pSet->IsEOF())
                m_pSet->MoveLast();
        if (m_pSet->IsBOF())
                m_pSet->SetFieldNull(NULL);

        UpdateData(FALSE);
}
```

Мы модифицировали приложение Employee, и теперь оно способно выполнять добавление и удаление записей, равно как и их обновление. Откомпилируйте приложение и запустите его на выполнение, выбрав в меню Visual Studio команду Build⇨Execute или нажав клавиши <Ctrl+F5>. Когда приложение Employee начнет работу, на экране раскроется его главное окно, внешний вид которого не претерпел никаких изменений в сравнении с предыдущей версией приложения. Однако теперь у вас появилась возможность добавить в базу данных новую запись, щелкнув на пиктограмме добавления записи на панели инструментов (или выбрав команду Record⇨Add Record), либо удалить текущую запись из базы, щелкнув на пиктограмме удаления записи на панели инструментов (или выбрав команду Record⇨Delete Record).

После щелчка на пиктограмме добавления записи приложение отображает в экранной форме пустую запись. Заполните поля новой записи. При переходе к другой записи приложение автоматически внесет новую запись в базу данных. Для того чтобы запись удалить, просто щелкните на пиктограмме удаления. Текущая запись (та, которая отображена на экране) исчезнет, и на экран будет выведена следующая запись базы данных.

Анализ функции OnRecordAdd()

Вероятно, вам будет интересно узнать, как работают подпрограммы на C++, добавленные в приложение.

Функция OnRecordAdd() начинает свою работу с вызова метода AddNew() класса CEmployeeSet, производного от класса CRecordset. Вызванная функция формирует пустую запись, предназначенную для заполнения пользователем. Однако эта запись не появится на экране до тех пор, пока не будет вызван метод UpdateData() класса представления. Но прежде чем осуществить вызов этого метода, необходимо еще кое-что сделать.

После того как пользователь создаст новую запись, необходимо будет обновить базу данных. Установка в данной подпрограмме определенного флажка позволит подпрограмме пересылки определить, какое именно действие пользователя имеет место: перемещение к следующей записи базы данных от существовавшей ранее записи базы или же от вновь добавленной. Именно с этой целью переменной m_bAdding присваивается значение TRUE.

В данный момент, когда пользователю предоставляется возможность ввести новую запись, необходимо изменить статус поля кода служащего Employee ID, обычно имеющего атрибут "только чтение". Для снятия этого атрибута программе прежде всего необходимо с помощью функции GetDlgItem() получить указатель на соответствующий элемент управления, а затем вызвать метод SetReadOnly() для присвоения значения FALSE атрибуту "только чтение" этого элемента управления.

Вот теперь все подготовлено к вызову функции UpdateData() для отображения на экране новой пустой записи.

Анализ функции OnMove()

Теперь, когда пустая запись выведена на экран, пользователю не составит большого труда заполнить поля ввода необходимыми данными. Для того чтобы новая запись действительно была помещена в базу данных, пользователю необходимо выполнить переход к другой записи базы. При этом будет вызван метод OnMove() класса представления. Обычно функция OnMove() не выполняет ничего, кроме отображения следующей записи базы данных. Сделанное нами переопределение этой функции дополнительно обеспечит и сохранение новой записи.

При вызове функция OnMove() прежде всего проверяет значение логической переменной m_bAdding и таким образом выясняет, от какой записи происходит переход: от существовавшей или от вновь добавленной. Если значение m_bAdding равно FALSE, то основное тело оператора if пропускается и выполняется фрагмент программы, следующий за else. При этом программа вызывает метод OnMove() базового класса (CRecordView), который выполняет обычный переход на следующую запись.

Если переменная m_bAdding имеет значение TRUE, выполняется основное тело оператора if. Здесь программа прежде всего сбрасывает флаг m_bAdding, а затем вызывает функцию UpdateData() для передачи данных из полей окна представления в буфер выбранных записей. Вызов функции CanUpdate() класса выборки данных определяет, можно ли обновлять источник данных, и, если можно, вызов функции Update(), являющейся членом этого же класса, добавляет новую запись к источнику данных.

Для формирования новой выборки данных программа должна вызвать функцию Requery(), являющуюся членом класса CRecordset, а затем вызовом метода класса окна представления UpdateData() поместить новые данные в элементы управления этого окна. И наконец, программа восстанавливает для поля кода служащего Employee ID атрибут “только чтение”, еще раз вызвав функции GetDlgItem() и SetReadOnly().

Анализ функции OnRecordDelete()

Удаление записи выполняется достаточно просто. Функция OnRecordDelete() вызывает функцию Delete(), являющуюся членом класса выборки данных. После выполнения удаления вызов метода MoveNext() класса выборки данных позволяет организовать переход к отображению следующей записи таблицы.

Однако здесь может возникнуть проблема, если удаляемая запись была в таблице последней или же единственной. Вызов метода IsEOF() класса CRecordset позволяет выяснить, достигнут ли конец последовательности записей. Если эта функция возвращает TRUE, то указатель записи нужно поместить на последнюю запись в текущей выборке. Для этого используется метод класса выборки данных MoveLast().

Когда все записи из текущей выборки данных будут удалены, указатель текущей записи будет находиться в начале выборки. Программа может проверить наличие такой ситуации посредством вызова метода IsBOF() класса CRecordset. Если эта функция возвращает TRUE, программа устанавливает значения полей текущей записи равными NULL.

Для завершения работы подпрограммы необходимо обновить содержимое окна представления, что осуществляется еще одним вызовом функции UpdateData().

Сортировка и фильтрация записей

Часто при работе с базой данных требуется изменить порядок, в котором записи отображаются на экране, или же осуществить поиск записей, удовлетворяющих определенному критерию. Существующие в MFC классы работы с базами данных ODBC располагают методами, позволяющими сортировать выбранные записи по любому из их полей. Кроме того, вызов

определенных методов этих классов предоставит возможность ограничить набор отображаемых записей только такими, поля которых содержат указанную информацию, например конкретное имя или идентификатор. Данная операция называется *фильтрацией*. В этом разделе мы добавим функции сортировки и фильтрации в приложение Employee. Выполните следующие действия.

1. Добавьте меню Sort (Сортировка) в основное меню приложения, как показано на рис. 22.29. Предоставьте Visual Studio автоматически определить идентификаторы команд.
2. С помощью мастера ClassWizard организуйте в классе CEmployeeView перехват четырех новых команд сортировки, используя имена функций, предложенные этим мастером. Окончательный вид окна ClassWizard показан на рис. 22.30.
3. Добавьте меню Filter (Фильтрация) в строку меню приложения, как показано на рис. 22.31. Предоставьте Visual Studio установить идентификаторы команд.
4. С помощью мастера ClassWizard организуйте в классе CEmployeeView перехват четырех новых команд фильтрации, используя имена функций, предложенные этим мастером.
5. Выберите команду Insert⇨Resource и создайте новое диалоговое окно, сделав двойной щелчок на элементе Dialog, а затем отредактируйте диалоговое окно так, чтобы оно выглядело, как показано на рис. 22.32. Присвойте элементу управления — текстовому полю — идентификатор ID_FILTERVALUE.
6. Оставив новое диалоговое окно раскрытым на экране, запустите мастер ClassWizard. Раскроется диалоговое окно Adding a Class. Установите опцию Create a new class и щелкните на кнопке OK.

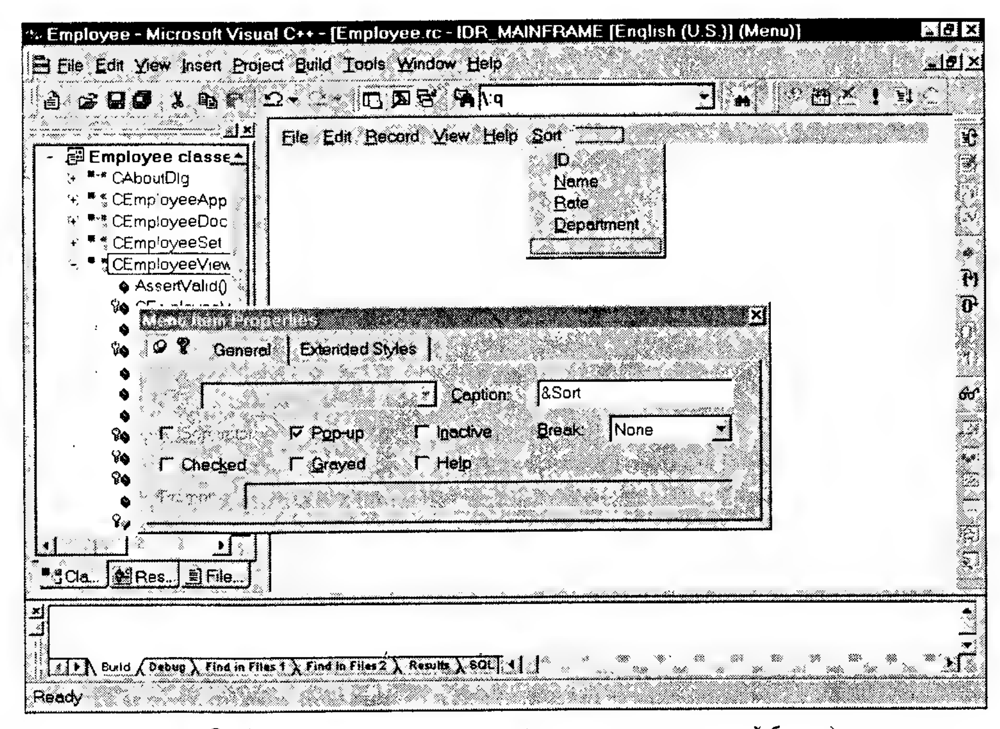


Рис. 22.29. Меню Sort *содержит четыре команды сортировки записей базы данных*

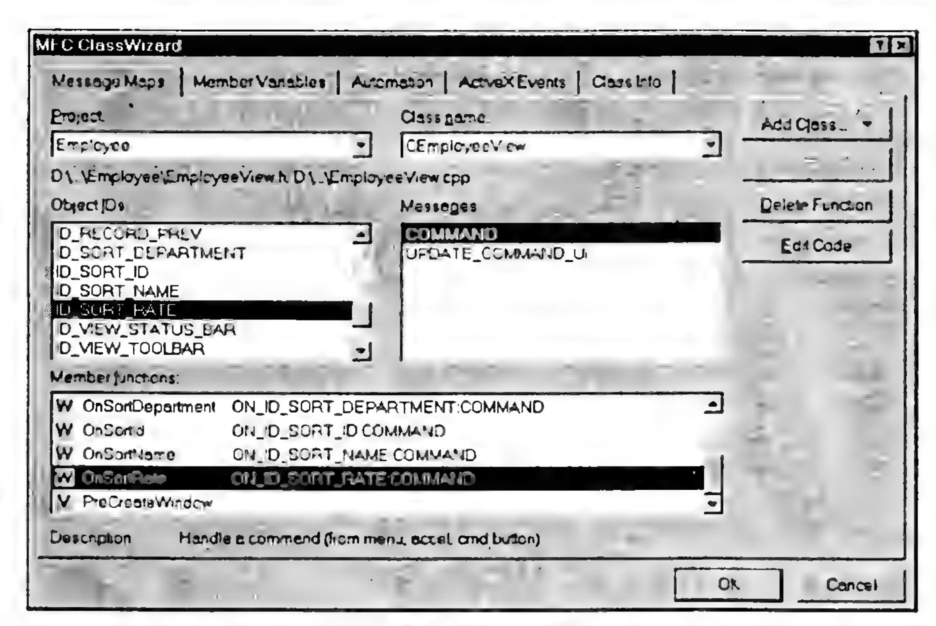


Рис. 22.30. Вид окна мастера ClassWizard после добавления четырех новых функций сортировки

Рис. 22.31. Меню Filter содержит четыре команды

Рис. 22.32. Создание диалогового окна установки параметров фильтрации

1. Раскроется диалоговое окно New Class. В поле Name введите значение CFilterDlg, как показано на рис. 22.33.
2. В окне мастера ClassWizard щелкните на корешке вкладки Member Variables. Свяжите элемент управления IDC_FILTERVALUE с переменной-членом m_filterValue. Завершите работу с мастером ClassWizard, щелкнув на кнопке OK.

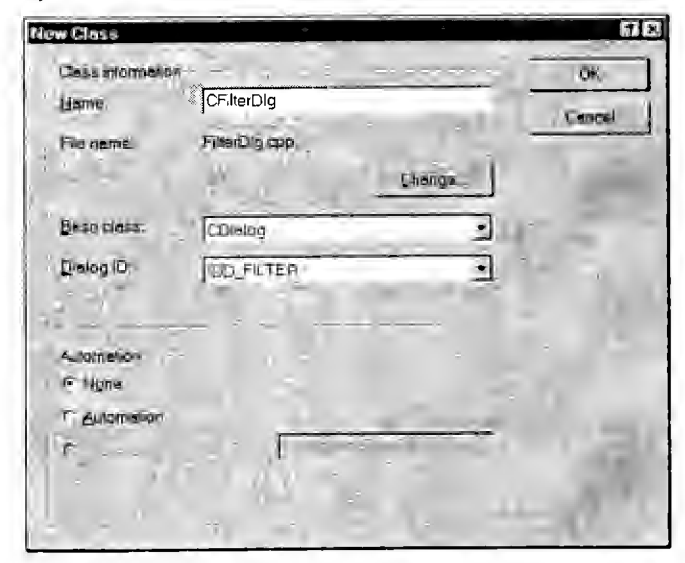


Рис. 22.33. Создание класса диалога для окна Filter

Теперь, когда меню и диалоговые окна уже созданы и связаны с заготовками функций, необходимо добавить в эти заготовки определенный программный код. На панели ClassView сделайте двойной щелчок на функции `OnSortDepartment()`, а затем отредактируйте ее текст в соответствии с листингом 22.4.

Листинг 22.4. Функция `CEmployeeView::OnSortDepartment()`

```
void CEmployeeView::OnSortDepartment()
{
        m_pSet->Close();
        m_pSet->m_strSort = "DeptID";
        m_pSet->Open();
        UpdateData(FALSE);
}
```

В окне ClassView сделайте двойной щелчок на функции `OnSortID()` и отредактируйте ее текст в соответствии с листингом 22.5. Повторите подобную операцию для функции `OnSortName()`, приведя ее текст в соответствие с листингом 22.6, и для функции `OnSortRate()`, текст которой должен соответствовать листингу 22.7.

Листинг 22.5. Функция `CEmployeeView::OnSortID()`

```
void CEmployeeView::OnSortId()
{
        m_pSet->Close();
        m_pSet->m_strSort = "EmployeeID";
        m_pSet->Open();
        UpdateData(FALSE);
}
```

Листинг 22.6. Функция `CEmployeeView::OnSortName()`

```
void CEmployeeView::OnSortName()
{
        m_pSet->Close();
        m_pSet->m_strSort = "EmployeeName";
        m_pSet->Open();
        UpdateData(FALSE):
}
```

Листинг 22.7. Функция `CEmployeeView::OnSortRate()`

```
void CEmployeeView::OnSortRate()
{
        m_pSet->Close();
        m_pSet->m_strSort = "EmployeeRate";
        m_pSet->Open();
        UpdateData(FALSE);
}
```

В начало файла `EmployeeView.cpp`, после уже имеющихся директив `#include`, добавьте следующую строку:

```
#include "FilterDlg.h"
```

Отредактируйте текст функций `OnFilterDepartment()`, `OnFilterID()`, `OnFilterName()`, `OnFilterRate()` в соответствии с листингом 22.8.

Листинг 22.8. Определение четырех функций фильтрации

```
void CEmployeeView::OnFilterDepartment()
{
	DoFilter("DeptID");
}

void CEmployeeView::OnFilterId()
{
	DoFilter("EmployeeID");
}

void CEmployeeView::OnFilterName()
{
	DoFilter("EmployeeName");
}

void CEmployeeView::OnFilterRate()
{
	DoFilter("EmployeeRate");
}
```

Все эти четыре функции вызывают функцию `DoFilter()`. Далее необходимо будет написать эту функцию, выполняющую фильтрацию записей базы данных, представленных в классе выборки данных. На панели ClassView щелкните правой кнопкой мыши на классе `CEmployeeView` и выберите в раскрывшемся контекстном меню команду Add Member Function. Укажите в раскрывшемся диалоговом окне тип функции `void` и введите ее объявление как `DoFilter(CString col)`. Данный метод должен быть защищенным, так как он вызывается только другими методами этого же класса `CEmployeeView`. На панели ClassView сделайте двойной щелчок на функции `DoFilter()` и поместите в нее текст программы, показанный в листинге 22.9.

Листинг 22.9. Функция `CEmployeeView::DoFilter()`

```
void CEmployeeView::DoFilter(CString col)
{
	CFilterDlg dlg;
	int result = dlg.DoModal();

	if (result == IDOK)
	{
		CString str = col + " = " + dlg.m_filterValue + "";
		m_pSet->Close();
		m_pSet->m_strFilter = str;
		m_pSet->Open();
		int recCount = m_pSet->GetRecordCount();

		if (recCount == 0)
		{
			MessageBox("No matching records.");
			m_pSet->Close();
			m_pSet->m_strFilter = "";
			m_pSet->Open();
		}

		UpdateData(FALSE);
	}

}
```

Мы добавили к создаваемому приложению функции сортировки и фильтрации записей базы данных. Оттранслируйте приложение и запустите его на выполнение. На экране появится главное окно приложения, которое выглядит так же, как и раньше. Однако теперь можно сортировать записи по любому полю, для чего достаточно просто выбрать имя поля в меню Sort. Кроме того, появилась возможность задать фильтрацию отображаемых записей, выбрав имя требуемого поля в меню Filter, а затем введя значение фильтра в раскрывшемся диалоговом окне Filter. Определить, как записи отсортированы или какой для них задан фильтр, вы сможете, перемещаясь от одной записи к другой. Попробуйте, например, отсортировать записи по отделам или по зарплате. А затем установите фильтрацию по любому из отделов, который вы видели, просматривая базу.

Анализ функции OnSortDept()

Все функции сортировки имеют одинаковую структуру. Они закрывают выборку данных, устанавливают свои переменные-члены m_strSort и снова открывают выборку данных, а затем вызывают функцию UpdateData() для обновления окна представления данными из вновь полученной отсортированной выборки данных. Однако в тексте функций сортировки вы не найдете ни одного вызова функции, в названии которой было бы слово Sort. Когда же в таком случае выполняется сортировка? Она выполняется, когда выборка данных открывается заново.

Объект класса CRecordset (как и объект любого другого класса, производного от CRecordset, например объект класса CEmployeeSet в этой программе) использует специальную строковую переменную m_strSort для определения способа упорядочения записей. Объект анализирует эту строковую переменную при формировании выборки данных и соответственно упорядочивает выбранные из базы записи.

Анализ функции DoFilter()

Всякий раз, когда пользователь выбирает команду из меню Filter, управляющая программа вызывает соответствующий этой команде метод: OnFilterDept(), OnFilterID(), OnFilterName() или OnFilterRate(). Каждая из этих функций ничего не делает, кроме вызова локального метода DoFilter(), передавая ему в качестве параметра строковую переменную, определяющую поле, по которому требуется выполнить фильтрацию.

Функция DoFilter() независимо от того, какая именно команда была выбрана в меню, всегда отображает одно и то же диалоговое окно, создавая экземпляр объекта класса диалогового окна и вызывая его метод DoModal().

Если значение result не равно IDOK, значит, пользователь выполнил щелчок на кнопке Cancel и весь оператор if пропускается, а функции DoFilter() остается только закончить свою работу.

Внутри конструкции if прежде всего создается строковая переменная, которая будет использоваться для фильтрации записей базы данных. Строковая переменная применяется для выполнения фильтрации записей так же, как это происходит при сортировке. В данном случае строковая переменная называется m_strFilter. Строка, которая используется для фильтрации записей базы данных, должна иметь следующий формат:

ИдентификаторПоля = Значение

Здесь *ИдентификаторПоля* является аргументом типа CString функции DoFilter(), а *Значение* вводится пользователем в диалоговом окне. Например, если пользователь выберет команду фильтрации по полю отдела и введет в диалоговом окне значение фильтра hardware, функция DoFilter() должна будет создать строку

```
DeptID = hardware
```

Сформировав указанную строку, программа будет готова к выполнению фильтрации записей. Для этого, как и в случае сортировки, выборка данных должна быть закрыта, а затем, при ее повторном открытии, функция DoFilter() выполнит формирование выборки данных с учетом требуемой фильтрации.

Что произойдет, если в результате работы установленного фильтра не будет выбрано ни одной записи? Хороший вопрос. Функция DoFilter() обнаруживает подобную ситуацию, подсчитывая количество записей в создаваемой выборке и сравнивая затем это число с нулем. Если набор записей пуст, программа выводит окно сообщения, информирующее пользователя о сложившейся ситуации. Затем программа закрывает выборку, присваивает строковой переменной фильтра пустое значение и снова открывает выборку записей. Таким образом восстанавливается выборка, включающая все записи таблицы Employees.

И наконец, независимо от того, удалось ли обнаружить записи, отвечающие заданному фильтру, или же выборка данных включает всю базу данных, программа должна заново отобразить данные на экране. Для этого вызывается функция UpdateData().

Выбор между классами OBDC и DAO

В предыдущем разделе вы получили общее представление о классах ODBC Visual C++ и узнали о том, как они используются в приложениях, созданных с помощью мастера AppWizard. Visual C++ также включает полный набор классов DAO, которые можно использовать для создания БД-приложений. DAO во многих отношениях является для классов ODBC суперклассом, включая большинство функциональных возможностей ODBC и добавляя при этом множество своих собственных. К сожалению, хотя классы DAO и могут работать с источниками данных ODBC, для которых существуют ODBC-драйверы, такое их применение не особенно эффективно. По этой причине DAO-классы больше подходят для создания программных приложений, оперирующих файлами баз данных формата .mdb фирмы Microsoft, создаваемых приложением Microsoft Access. Файлы других форматов, с которыми можно, используя классы DAO, работать напрямую, создаются приложениями FoxPro и Excel.

Классы DAO, которые используют приложение Microsoft Jet Database Engine, настолько похожи на классы ODBC, что во многих случаях можно конвертировать программы ODBC в DAO путем простого изменения названия класса в тексте программы: CDatabase изменяется на CDaoDatabase, CRecordset изменяется на CDaoRecordset, а CRecordView изменяется на CDaoRecordView. Однако между классами ODBC и DAO имеется существенное различие в том, как реализуются системные библиотеки. ODBC-классы реализованы как набор модулей DLL, в то время как классы DAO реализованы в виде объектов OLE. Использование объектов OLE делает систему DAO несколько более современной в сравнении с ODBC, по крайней мере в отношении архитектуры.

Хотя система DAO реализована в виде объектов OLE, вам не придется беспокоиться о работе с подобными объектами напрямую. Входящие в MFC классы DAO берут обработку всех деталей управления на себя, предоставляя данные и методы, обеспечивающие взаимодействие с объектами OLE. Класс CDaoWorkspace обеспечивает с помощью статических методов прямой доступ к объектам ядра базы данных DAO. Хотя MFC берет управление рабочей областью на себя, можно использовать ее данные и методы для непосредственной инициализации связи с базой данных.

Еще одно отличие состоит в том, что классы DAO предоставляют более мощный набор функций, которые можно использовать для манипулирования базой данных. Эти более мощные методы позволяют выполнять с базами данных сложные операции, используя небольшой объем исходного текста на C++ или SQL-выражения, написанные непосредственно разработчиком.

- Обе системы (ODBC и DAO) могут работать с ODBC-источниками данных. Однако DAO менее эффективна при таком применении, так как больше подходит для работы с файлами баз данных формата .mdb.
- Мастер AppWizard может создать заготовку БД-приложения, используя либо классы ODBC, либо классы DAO. Выбор типа создаваемого приложения зависит, по крайней мере частично, от баз данных, с которыми вы будете работать.
- Обе системы — и ODBC, и DAO — используют для соединения с базой данных, к которой осуществляется доступ, объекты классов баз данных MFC. В ODBC такой класс базы данных называется CDatabase, а в системе DAO — CDaoDatabase. Хотя эти классы и имеют разные названия, DAO-класс содержит множество членов, подобных тем, которые можно обнаружить в ODBC-классе.
- Обе системы, ODBC и DAO, используют объекты класса выборки данных для хранения записей, выбранных на текущий момент. В ODBC такой класс выборки данных называется CRecordset, а в системе DAO — CDaoRecordset. Хотя эти классы и имеют разные названия, DAO-класс выборки данных содержит практически все члены класса ODBC. Кроме того, DAO-класс имеет большой набор дополнительных методов.
- Снстемы ODBC и DAO используют схожие методики просмотра содержнмого нсточника данных, а именно: в обеих системах приложенне должно создать объект базы данных, создать объект выборки данных, а затем вызвать методы соответствующего класса для манипулирования базой данных.

Различия между системами ODBC и DAO состоят в следующем.

- Хотя входящие в MFC классы ODBC и DAO похожи (иногда даже очень), некоторые аналогичные методы имеют разные имена. Кроме того, классы DAO включают много методов, которым нет аналогов в классах ODBC.
- В системе ODBC для определения опций, которые могут использоваться при открытии выборки данных, используются макросы и перечисления, тогда как в DAO для этих целей определены константы.
- Большое количество существующих ODBC-драйверов делает систему ODBC пригодной для работы с множеством файлов баз данных различных форматов, в то время как система DAO больше подходит для приложений, работающих только с файлами формата .mdb.
- Система ODBC реализована в виде набора DLL-модулей, а DAO реализована как набор объектов OLE.
- В ODBC объект класса CDatabase напрямую взаимодействует с источником данных. В DAO объект класса CDaoWorkspace занимает промежуточное положение между объектами классов CDaoRecordset и CDaoDatabase, что дает возможность рабочей среде взаимодействовать со многими объектами класса баз данных.

OLE DB

OLE DB — это совокупность интерфейсов OLE (ActiveX), которые упрощают доступ к данным, сохраненным приложениями, не являющимися СУБД, например хранящимся в почтовых ящиках электронной почты или линейных файлах. Приложение, использующее OLE DB, может интегрировать информацию из таких СУБД, как Oracle, SQL Server и Access, с информацией из систем, не являющихся СУБД, но использующих возможности OLE (ActiveX).

Предоставление полного руководства по применению OLE DB выходит далеко за рамки данной главы. Для использования этого мощного инструмента вы должны уверенно чувство-

вать себя при работе с интерфейсами OLE. Если вам раньше приходилось создавать приложения OLE (ActiveX) только с помощью MFC и мастера AppWizard, вас может шокировать знакомство с тем, что Microsoft полагает "упрощенным". Вы встретитесь с множеством вызовов функции QueryInterface() и множеством переменных с именами наподобие pIColsInfo и rgColInfo. Все же, разобравшись во всех интерфейсах и всех установках, вы сможете вызвать такую функцию, как GetData(), и получить информацию из приложения, не являющегося СУБД, так, как будто это база данных. В результате получается существенный выигрыш во времени.

В электронной документации по Visual C++ есть руководство *OLE DB Programmer's Reference*. Если вы хорошо знакомы с концепциями OLE и ActiveX, то было бы неплохо начать именно с этого руководства.

ГЛАВА 23

SQL и редакция Visual C++ Enterprise Edition

В этой главе...

Особенности редакции Enterprise Edition

Что такое SQL

Базы данных SQL и C++

Анализ приложения для издательства

Работа с базой данных

Что такое Microsoft Transaction Server

Использование Visual SourceSafe

Особенности редакции Enterprise Edition

Редакция Enterprise Edition пакета Visual C++ была разработана для тех программистов, которые создают на C++ приложения, использующие язык SQL для работы с базами данных и в особенности при применении хранимых процедур. Вы можете приобрести Visual C++ в редакции Enterprise Edition вместо Professional Edition. Если у вас уже есть редакция Professional или Subscription Edition, можете приобрести Enterprise Edition со скидкой.

В состав редакции Enterprise Edition пакета Visual C++ включены некоторые дополнительные возможности.

- Отладка выражений на языке SQL
- Мастер отладки хранимых процедур Extended Stored Procedure Wizard
- Поддержка OLE DB для доступа к AS 400

Кроме того, в пакет включены дополнительные инструментальные средства.

- Visual SourceSafe
- SQL Server 6.5 (developer Edition, SP 3)
- Visual Modeler
- Microsoft Transition Server
- Internet Information Server 4.0

Что такое SQL

SQL (Structured Query Language — язык структурированных запросов) является средством доступа к базам данных и предназначен для использования в интерактивном или программном режиме. Его грамматический строй максимально приближен к строю обычного английского языка. Большинство SQL-выражений являются *запросами* (query) выборки информации из одной или нескольких баз данных. Кроме того, в SQL предусмотрены конструкции для добавления, удаления и изменения информации. Как уже упоминалось в главе 22, язык SQL — тема для отдельной книги. В данном разделе дается краткий обзор наиболее важных команд SQL, так что вы получите представление о мощности и богатстве возможностей этого инструмента и сможете разобраться в приведенных примерах, даже если раньше не использовали SQL.

Язык SQL используется для доступа к реляционным базам данных, содержащим некоторое количество *таблиц*. Таблица состоит из строк, строки — из столбцов. В табл. 23.1 перечислены термины, используемые в качестве синонимов для понятий "таблица", "строка" и "столбец" в научной литературе по теории баз данных и в документации по некоторым другим типам баз данных.

Таблица 23.1. Терминология баз данных

SQL	Существующие синонимы
Таблица (Table)	Сущность (Entity)
Строка (Row)	Запись, кортеж (Record, Tuple)
Столбец (Column)	Поле, атрибут (Field, Attribute)

Вот пример предложения на языке SQL:

```
SELECT au_fname, au_lname FROM authors
```

Результатом выполнения этого запроса будет список имен и фамилий авторов, информация о которых хранится в таблице с именем authors. (Эта таблица входит в состав включенной в поставку SQL Server учебной базы данных pubs, используемой в примерах данной главы.) Ниже приводится более сложное SQL-выражение.

```
SELECT item, SUM(amount)total, AVG(amount) average FROM ledger
       WHERE action = PAID
       GROUP BY item
having AVG(amount) > (SELECT avg(amount) FROM ledger
                            WHERE action = PAID)
```

SQL-выражение состоит из ключевых слов, имен таблиц и имен столбцов. К ключевым словам относятся следующие.

- SELECT. Возвращает определенный столбец из базы данных. К уточняющим ключевым словам этой команды, ограничивающим поиск до определенной записи в каждой таблице, относятся FROM, WHERE, LIKE, NULL и ORDER BY.
- DELETE. Удаляет записи. Уточняющее ключевое слово WHERE специфицирует удаляемую запись.
- UPDATE. Изменяет значения столбцов (специфицированных с помощью фразы SET) в записях, определяемых с помощью фразы WHERE. Может комбинироваться с командой SELECT.
- INSERT. Помещает в базу данных новую запись.
- COMMIT. Фиксирует любые сделанные ранее в базе данных изменения.
- ROLLBACK. Аннулирует все изменения, сделанные со времени последнего применения команды COMMIT.
- EXEC. Вызывает на выполнение хранимую процедуру.

Подобно языку C++, в SQL поддерживается два вида комментариев:

```
/*Этот комментарий содержит открывающие и закрывающие символы */
- Этот комментарий начинается здесь и заканчивается в конце строки.
```

Базы данных SQL и C++

Как мы уже знаем из главы 22, программы на C++ с помощью классов ODBC CDatabase и CRecordset могут работать и с базой данных SQL Server или любой другой СУБД, поддерживающей запросы на SQL. Больше того, метод ExecuteSQL() класса CDatabase дает возможность прямо из программы выполнить любую команду языка SQL. В большинстве случаев выполняемая команда SQL будет вызовом одной из хранимых процедур — некоторой последовательности команд SQL, хранимых совместно с базой данных и предназначенных для выполнения определенной операции обработки данных.

Существует достаточно причин, по которым не следует помещать текст команд SQL непосредственно в программы. Тремя наиболее существенными из них являются следующие.

- Повторное использование
- Разделение по специализации
- Удобство сопровождения (модифицируемость)

Многие программисты при написании на C++ приложений для работы с базами данных на протяжении многих лет используют одни и те же хранимые процедуры, подготовленные другими разработчиками при создании определенной базы данных. Копировать эти процедуры непосредственно в текст С-программ, по крайней мере, неразумно. В то же время обращение к ним изнутри программы обеспечивает удобные пользовательские интерфейсы, упрощает доступ к Internet и позволяет обеспечить высокую эффективность, присущую программам, написанным на C++, сохранив при этом всю мощь написанных ранее хранимых процедур.

Высококвалифицированные специалисты требуются всегда, и случается, что спрос на них превышает предложение. Многие компании испытывают трудности в поисках опытных программистов на C++, а также квалифицированных администраторов баз данных, которые могли бы изучить структуру базы данных и работать с ней на SQL. Представьте себе, как сложно найти человека, который смог бы удовлетворять и тому, и другому критериям одновременно. Почти так же трудно, как ожидать от двух разработчиков одновременной работы над различными частями одной и той же программы, в которой обращение к SQL выполняется в исходном тексте на C++. Поэтому гораздо удобнее иметь программиста на C++, умеющего реализовать вызов хорошо документированных хранимых процедур, и разработчика на SQL, создающего такие хранимые процедуры и обеспечивающего нормальное функционирование базы данных.

Разбиение приложения на независимые части C++ и SQL имеет еще одно преимущество. Изменения, выполняемые в одной из частей, могут не оказывать влияния на другую. Например, меньшие по размеру программы на C++, не содержащие в себе текста команд SQL, будут компилироваться и компоноваться гораздо быстрее именно за счет уменьшения их размеров. Изменения в хранимых процедурах SQL, если они не затрагивают аргументов функций или возвращаемых ею значений, не потребуют повторного компилирования и компоновки программы на C++.

Однако этому подходу присущи и определенные недостатки. Очень трудно выявить причины возникновения некоторой проблемы, если вы заранее не знаете точно, в какой из частей программы их искать — C++ или SQL. С другой стороны, если один и тот же разработчик работает над обеими частями, необходимость изучения двух различных инструментов и выполнения постоянных переключений между ними в процессе работы, безусловно, усложняет разработку в сравнении с использованием одного инструмента. Кроме того, существующие сегодня инструменты работы с SQL не имеют многих возможностей, предоставляемых программистам, работающим на Visual C++.

В настоящий момент редакция Enterpise Edition пакета Visual C++ предоставляет в ваше распоряжение самые лучшие инструменты для обеих частей приложения такого рода. Вы можете в целях удобства сопровождения и получения возможности многократного использования разделить приложение на части C++ и SQL и при этом пользоваться редактором, обеспечивающим синтаксическую раскраску, и даже отладчиком Visual C++ для работы с хранимыми процедурами SQL.

Анализ приложения для издательства

Один из примеров базы данных, поставляемый вместе с SQL Server, называется pubs. Эта база предназначена для учета продажи книг и выплаты гонораров их авторам. В данной главе мы создадим новую хранимую процедуру и осуществим вывод данных, выбранных из базы с ее помощью, в простое диалоговое окно просмотра записей. Внимание! SQL Server должен быть установлен и запущен до того, как мы приступим к созданию данного приложения.

Установка источника данных

Прежде чем приступать к созданию проекта, нужно сформировать источник данных, к которому приложение будет обращаться. В реальном проекте этот источник данных должен уже существовать к моменту начала работы над проектом.

Из меню Start в Windows 95 откройте Control Panel. В ее окне сделайте двойной щелчок на пиктограмме ODBC. Раскроется диалоговое окно ODBS Data Source Administrator, показанное на рис. 23.1. На вкладке User DSN щелкните на кнопке Add и добавьте имя нового источника данных (DSN — data source name).

В диалоговом окне Create New Data Source (Создать новый источник данных), которое появится в результате этого на экране, выберите из списка драйверов SQL Server, как показано на рис. 23.2, а затем щелкните на кнопке Finish (Готово).

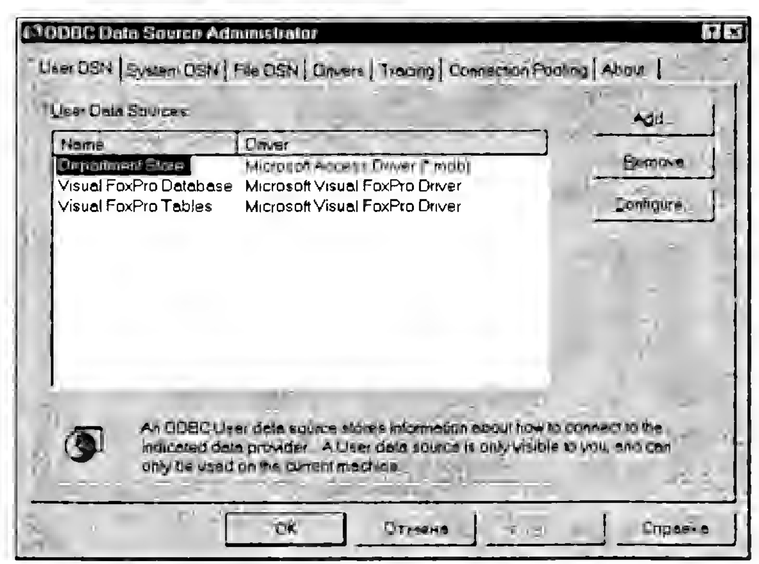


Рис. 23.1. Добавление имени нового источника данных

Create New Data Source

Select a driver for which you want to set up a data source

Name	Version	Company
Microsoft Access Driver (*.mdb)	3.51.171300	Microsoft Corporation
Microsoft dBase Driver (*.dbf)	3.51.171300	Microsoft Corporation
Microsoft Excel Driver (*.xls)	3.51.171300	Microsoft Corporation
Microsoft FoxPro Driver (*.dbf)	3.51.171300	Microsoft Corporation
Microsoft ODBC Driver for Oracle	2.00.006325	Microsoft Corporation
Microsoft ODBC for Oracle	2.573.292700	Microsoft Corporation
Microsoft Paradox Driver (*.db)	3.51.171300	Microsoft Corporation
Microsoft Text Driver (*.txt, *.csv)	3.51.171300	Microsoft Corporation
Microsoft Visual FoxPro Driver	6.00.816700	Microsoft Corporation
SQL Server	3.60.0319	Microsoft Corporation

Готово Отмена

Рис. 23.2. Подключение сервера SQL

В следующем диалоговом окне, Create a New Data Source to SQL Server, заполните поля Name (Имя источника данных) и Description (Описание). Затем раскройте список Server и выберите из него подходящий сервер или введите его имя самостоятельно. На рис. 23.3 показано диалоговое окно, все поля которого заполнены. Теперь щелкните на Next (Далее).

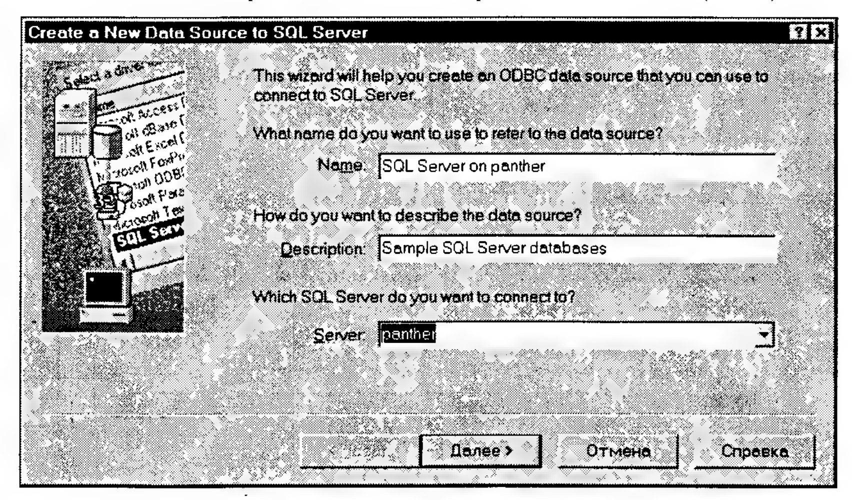


Рис. 23.3. Спецификация сервера

При подключении к серверу можно использовать параметры регистрации как в операционной системе NT, так и в среде обслуживания SQL серверов. Если в процедуре регистрации вам что-либо покажется непонятным, проконсультируйтесь у системного администратора. Для примера, который будет рассмотрен дальше, вполне достаточно было зарегистрироваться в среде обслуживания SQL серверов, причем даже без пароля.

На этом этапе нужно решить, будете ли вы подключаться к определенной базе данных на сервере или к серверу в целом. Если формируемое DSN должно быть связано с определенной базой, установите самый верхний флажок в окне и выберите в расположенном под ним поле интересующую вас БД. В противном случае оставьте этот флажок сброшенным. В любом случае остальные элементы управления на этой странице оставьте в том виде, в котором предлагает система.

В следующем диалоговом окне оставьте установки параметров, предлагаемые по умолчанию, и щелкните на Next.

В последнем диалоговом окне настройки оставьте оба флажка сброшенными. Щелкните на Finish (Готово) — и процесс на этом завершится.

Щелкните на кнопке Test Data Source (Проверка источника данных). Если результат тестирования вас не удовлетворит, щелкните на Cancel и вернитесь к последнему окну настройки, а затем, щелкая на Back, перейдите к тому окну, в котором нужно откорректировать параметры.

Когда тестирование вас удовлетворит, щелкните на OK в итоговом окне, а затем еще и на OK в окне ODBC Source Administrator. Закройте Control Panel.

Создание оболочки приложения

Запустите Visual Studio и выберите команду File⇨New, а затем в открывшемся диалоговом окне выберите вкладку Projects. Выберите в списке значение `MFC AppWizard (exe)` и присвойте проекту имя `Publishing` (Издательство), как показано на рис. 23.4. Для запуска `AppWizard` щелкните на OK.

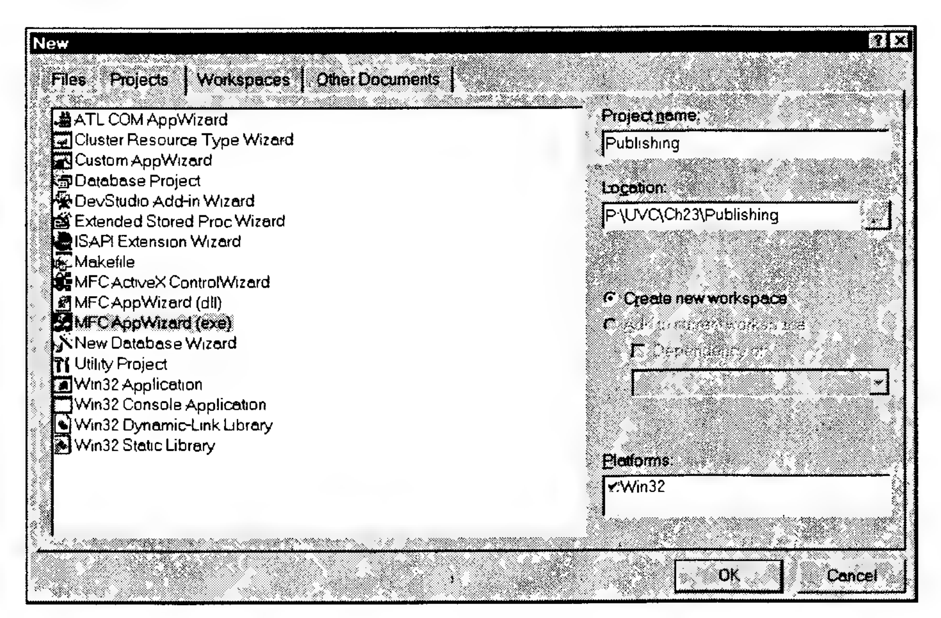


Рис. 23.4. Запустите AppWizard как обычно

На первом этапе настройки AppWizard выберите приложение типа SDI. Для перехода ко второму этапу настройки AppWizard щелкните на кнопке Next. Установите переключатель Database view without file support, как показано на рис. 23.5. Чтобы соединить создаваемое приложение с источником данных, щелкните на кнопке Data Source.

Установите в группе Datasource переключатель ODBC и выберите в раскрывающемся списке имя того источника данных, который был подсоединен в результате выполнения операций, описанных выше, как показано иа рис. 23.6. В группе Recordset type (Тип выборки даниых) оставьте установку Snapshot и щелкните на кнопке OK, переходя к точному определению источника данных.

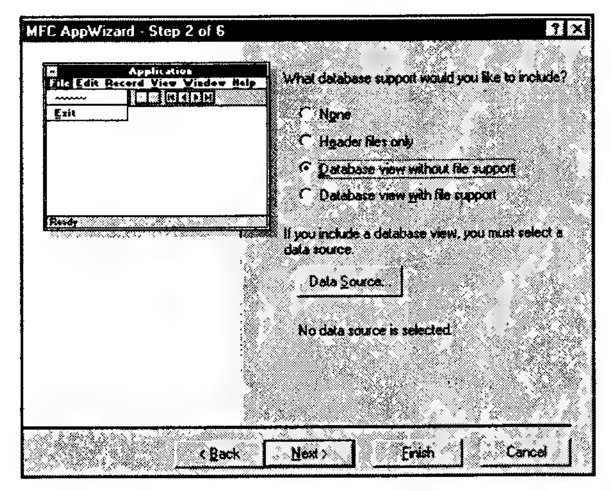


Рис. 23.5. Приложение будет работать с базой данных, но не будет создавать собственных документов

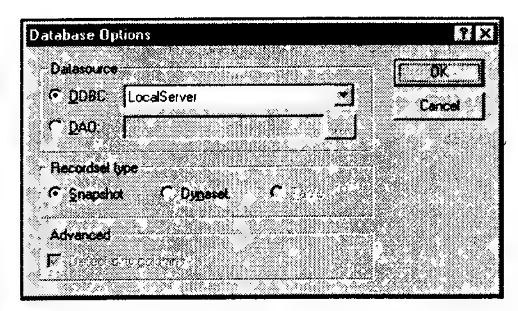


Рис. 23.6. Источником данных будет база данных ODBC

Раскроется диалоговое окно SQL Server Login (Подключение к SQL Server). Для вывода на экран расширенного диалогового окна, показанного на рис. 23.7, щелкните на кнопке Options. В раскрывающемся списке Database выберите значение `pubs` и введите в верхней части диалогового окна ваш идентификатор и пароль для входа в систему. Щелкните на OK.

Раскроется диалоговое окно Select Database Tables (Выбор таблиц базы данных), показанное на рис. 23.8. Выберите таблицы `dbo.authors`, `dbo.titleauthor` и `dbo.titles`, а затем щелкните на OK.

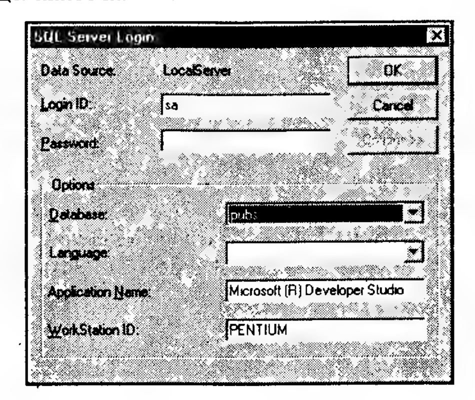


Рис. 23.7. Подключение к базе данных `pubs`

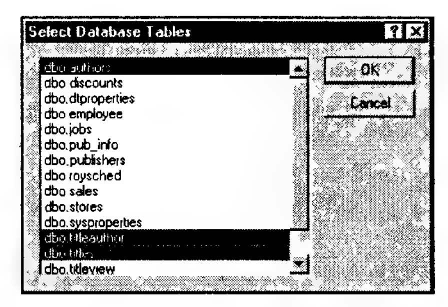


Рис. 23.8. Выбор таблиц `authors`, `titleauthor` и `titles`

Мы снова вернулись в окно второго этапа настройки `AppWizard`. Для перехода к третьему этапу щелкните на кнопке Next. Сбросьте флажки опций поддержки составных документов и элементов управления ActiveX, а затем щелкните на кнопке Next для перехода к четвертому этапу. Для принятия установок, предложенных в этом окне по умолчанию, щелкните на кнопке Next. На пятом этапе настройки повторите аналогичную процедуру. На шестом этапе щелкните на кнопке Finish. Раскроется окно New Project Information, содержащее сводку выполненных установок. Для запуска процедуры генерации проекта щелкните на OK.

Итак, завершен процесс создания оболочки приложения, которое будет отображать содержимое базы данных в окне просмотра записей почти точно так, как и приложение, созданное нами в главе 22. Однако до сих пор выполненные действия не содержали ничего, специфичного именно для `Enterprise Edition`.

Установление связи с данными

Выбранные ранее таблицы базы данных уже связаны с классом выборки записей, но к ним еще нельзя применять существующие в Enterprise Edition функции SQL. Необходимо установить связь с данными, что обеспечит взаимодействие базы данных с приложением. Для установления такой связи выполните следующие действия.

1. Выберите команду Project⇨Add to Project⇨New.
2. В раскрывшемся диалоговом окне перейдите на вкладку Projects.
3. В списке выберите значение Database Project, как показано на рис. 23.9, присвойте проекту имя PubDB и установите переключатель Add to current workspace. Щелкните на кнопке OK.
4. Раскроется диалоговое окно Select Data Source. Выберите значение Local Server, как показано на рис. 23.10, а затем щелкните на кнопке OK.
5. На экране вновь появится диалоговое окно SQL Server Login. Как и в предыдущем случае, укажите ваш идентификатор и пароль для входа в систему, а также убедитесь, что выбрана именно база данных pubs. Для завершения процедуры установления связи с данными щелкните на кнопке OK.

В окне Workspace в левой части экрана появилась вкладка DataView, которая показана на рис. 23.11. Раскройте элемент Tables, а в нем раскройте таблицу Authors, отобразив ее содержимое. Сделайте на таблице Authors двойной щелчок, и в правой части окна будут отображены хранящиеся в ней данные (рис. 23.11).

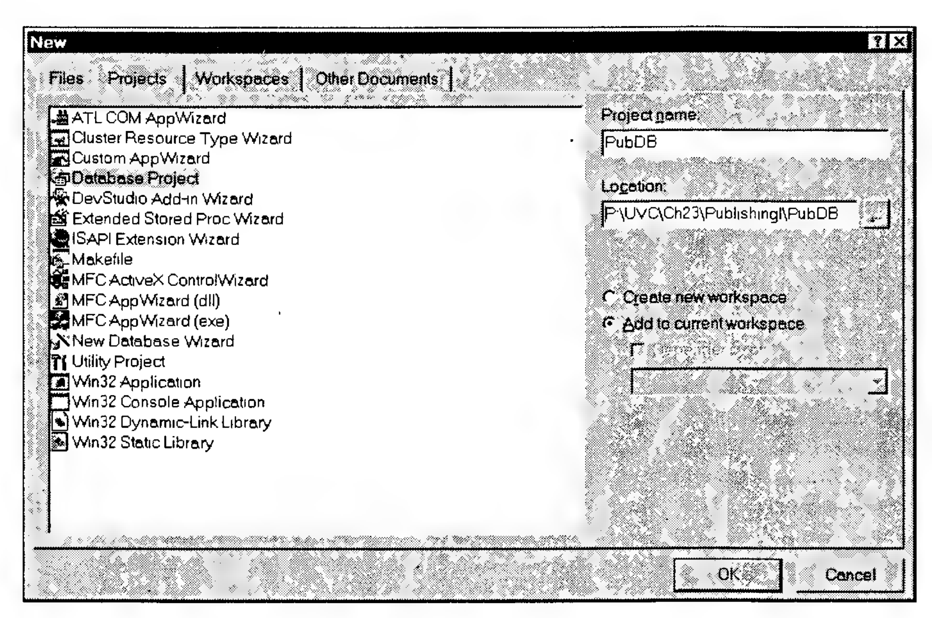


Рис. 23.9. Создание субпроекта в составе основного проекта

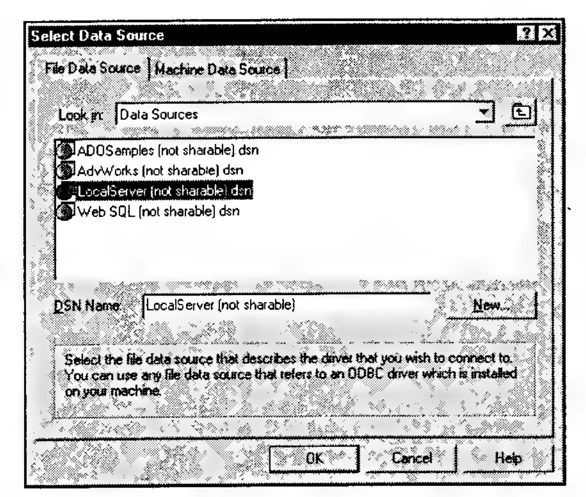


Рис. 23.10. Подключение к локальному серверу

au_id	au_lname	au_fname	phone	address
172-32-117	White	Johnson	408 496-7223	10932 Bigge Rd.
213-46-8915	Green	Marjorie	415 986-7020	309 63rd St. #411
238-95-7766	Carson	Cheryl	415 548-7723	589 Darwin Ln.
267-41-2394	O'Leary	Michael	408 286-2428	22 Cleveland Av. #
274-80-9391	Straight	Dean	415 834-2919	5420 College Av.
341-22-1782	Smith	Meander	913 843-0462	10 Mississippi Dr.
409-56-7008	Bennet	Abraham	415 658-9932	6223 Bateman St.
427-17-2319	Dull	Ann	415 836-7128	3410 Blonde St.
472-27-2349	Gringlesby	Burt	707 938-6445	PO Box 792
486-29-1786	Locksley	Charlene	415 585-4620	18 Broadway Av.
527-72-3246	Greene	Mornings	615 297-2723	22 Graybar House
648-92-1872	Blotchet-Hall:	Reginald	503 745-6402	55 Hillsdale Bl.
672-71-3249	Yokomoto	Akiko	415 935-4228	3 Silver Ct.
712-45-1867	del Castillo	Innes	615 996-8275	2286 Cram Pl. #86
722-51-5454	DeFrance	Michel	219 547-9982	3 Balding Pl.
724-08-9931	Stringer	Dirk	415 843-2991	5420 Telegraph Av
724-80-9391	MacFeather	Stearns	415 354-7128	44 Upland Hts.
756-30-7391	Karsen	Livia	415 534-9219	5720 McAuley St.
807-91-6654	Panteley	Sylvia	301 946-8853	1956 Arlington Pl.
846-92-7186	Hunter	Sheryl	415 836-7128	3410 Blonde St.
893-72-1158	McBadden	Heather	707 448-4982	301 Putnam
899-46-2035	Ringer	Anne	801 826-0752	67 Seventh Av.
998-72-3567	Ringer	Albert	801 826-0752	67 Seventh Av.

Рис. 23.11. На вкладке DataView *показана структура базы данных, а в рабочем окне отображены данные из таблицы* `Authors`

На рис. 23.11 также показана панель инструментов Query, на которой расположены следующие пиктограммы.

- Show Diagram Pane — выводит на экран панель диаграмм `Query Designer`, которая обсуждается в следующем разделе.
- Show Grid Pane — выводит на экран панель таблиц `Query Designer`, которая обсуждается в следующем разделе.

- Show SQL Pane — выводит на экран панель SQL Query Designer, которая обсуждается в следующем разделе.
- Show Results Pane — выводит на экран и/или удаляет с экрана панель результатов Query Designer, которая обсуждается в следующем разделе.
- Change Type — формирует на четырех панелях Query Designer запросы SELECT, INSERT, UPDATE и DELETE.
- Run — выполняет команду SQL.
- Verify SQL Syntax — проверяет синтаксис указанной команды SQL.
- Sort Ascending — отображает записи в указанном столбце в порядке от меньшего значения к большему.
- Sort Descending — отображает записи в указанном столбце в порядке от большего значения к меньшему.
- Remove Filter — отображает все записи, отменяя фильтрацию.
- Properties — отображает информацию о столбце или таблице.

Работа с Query Designer

Для того чтобы вывести на экран все столбцы всех записей некоторой таблицы, нужно сделать двойной щелчок на имени этой таблицы, например authors, в окне DataView в левой части экрана среды разработки. При этом фактически формируется простой запрос SQL, текст которого показан ниже:

```
SELECT authors.* FROM authors
```

Результат выполнения запроса выводится на панель результатов — единственную из четырех панелей Query Designer, которая выводится на экран по умолчанию. Приведенный выше запрос построен самим Query Designer и означает *отобразить все столбцы и записи таблицы* authors. На рис. 23.12 показаны все четыре панели окна Query Designer в том виде, который они имеют, когда впервые выполняется подключение к данным. Управление отображением этих панелей (установка-удаление панелей на экране) осуществляется с помощью пиктограмм панели инструментов. Можно настроить высоту каждой из панелей, но ширина их останется неизменной.

Для модификации этого запроса сбросьте на панели диаграмм (показана в верхней части рис. 23.12) флажок выборки элемента * (All Columns), а затем установите флажки выборки колонок au_lname, au_fname и au_pfone. Данные на панели результатов будут выведены затененными, указывая на то, что они устарели — не являются результатом формируемого сейчас запроса. По мере того, как вносятся изменения на панели диаграммы, содержимое остальных панелей (кроме панели результатов) обновляется, как показано на рис. 23.13.

Выделите на панели диаграмм поле phone и щелкните на пиктограмме Sort Ascending панели инструментов Query. Тем самым будет заказана сортировка результатов запроса по номеру телефона. После щелчка на пиктограмме Run панели инструментов Query автоматически сформированный запрос будет выполнен. На рис. 23.14 показан результат выполнения созданного запроса на обновленной панели результатов.

Рис. 23.12. Структура базы данных представлена в окне DataView, *а в рабочей области отображаются все четыре панели окна* Query Designer

Рис. 23.13. Простые запросы можно построить, даже не зная языка SQL

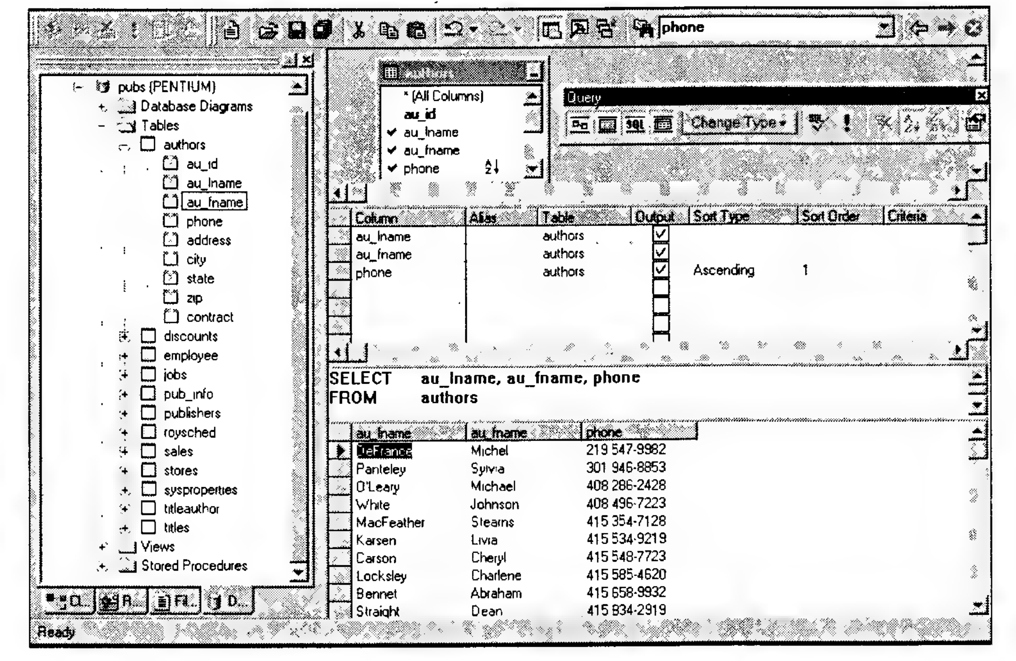


Рис. 23.14. Щелкнув мышью на пиктограмме Run, можно запустить на выполнение запрос SQL

Хранимые процедуры

Возможность быстро создавать простые запросы SQL, даже не имея навыков работы с этим языком, является замечательным качеством Enterprise Edition. Однако истинная ценность этого программного продукта заключается в возможности использования хранимых процедур.

Сверните в окне DataView ветвь таблиц и разверните вместо нее ветвь Stored Procedures (Хранимые процедуры). В этом разделе будут отображены все хранимые процедуры, помещенные в выбранную базу данных и готовые к применению. Дважды щелкните на элементе reptq2, и на экран будет выведен текст этой процедуры.

Едва ли вы оставите без внимания выделение цветом синтаксических элементов запросов в окне редактора. С этой целью используются следующие цвета:

- **синий** — для ключевых слов, например PRINT или SELECT;
- **зеленый** — для комментариев обоих типов;
- **черный** — для всего остального текста.

Для выполнения хранимой процедуры выберите команду Tools⇨Run или же щелкните правой кнопкой мыши на имени нужной процедуры в окне DataView, а затем выберите в раскрывшемся контекстном меню команду Run. Еще один вариант — щелкните правой кнопкой мыши в окне редактора и выберите команду Run в раскрывшемся контекстном меню. Результаты выполнения хранимой процедуры появятся на панели Results в окне Output (в нижней части экрана среды разработки); не путайте эту панель с панелью Results окна Query Designer. На рис. 23.15 показано окно Output, максимально развернутое для отображения результатов выполнения процедуры reptq2.

```
CREATE PROCEDURE reptq2 AS
select type, pub_id, titles.title_id, au_ord,
   Name = substring (au_lname, 1,15), ytd_sales
from titles, authors, titleauthor
where titles.title_id = titleauthor.title_id
   AND authors.au_id = titleauthor.au_id
   AND pub_id is NOT NULL
order by pub_id, type
COMPUTE avg(ytd_sales) BY pub_id, type
COMPUTE avg(ytd_sales) BY pub_id
```

```
Running dbo.reptq2.
type          pub_id title_id au_ord Name            ytd_sales
------------- ------ -------- ------ --------------- -----------
business      0736   BU2075   1      Green           18722
(1 row(s) affected)

AVG
-----------
18722
(1 row(s) affected)

type          pub_id title_id au_ord Name            ytd_sales
------------- ------ -------- ------ --------------- -----------
psychology    0736   PS2091   2      Ringer          2045
psychology    0736   PS2091   1      Ringer          2045
psychology    0736   PS2106   1      Ringer          111
psychology    0736   PS3333   1      White           4072
psychology    0736   PS7777   1      Locksley        3336
(5 row(s) affected)

AVG
```

Рис. 23.15. В окне Visual Studio можно увидеть результаты выполнения любой хранимой процедуры

Некоторым хранимым процедурам передаются параметры. Чтобы увидеть пример такого варианта, откройте текст процедуры reptq3, выполнив двойной щелчок на ее имени.

```
CREATE PROCEDURE reptq3 @lolimit money, @hilimit money,
@type char (12)
AS
select pub_id, type, title_id, price
from titles
where price >@lolimit AND price <@hilimit AND type = @type
      OR type LIKE %cook%
order by pub_id, type
COMPUTE count(title_id) BY pub_id, type
```

Этой хранимой процедуре передаются три параметра — lolimit (нижняя граница), hilimit (верхняя граница) и type (тип). Когда вы ее запустите, на экран будет выведено диалоговое окно, показанное на рис. 23.16. Введите значения параметров в правой колонке Value и щелкните на кнопке OK для продолжения выполнения процедуры. Результат вы увидите в окне Output.

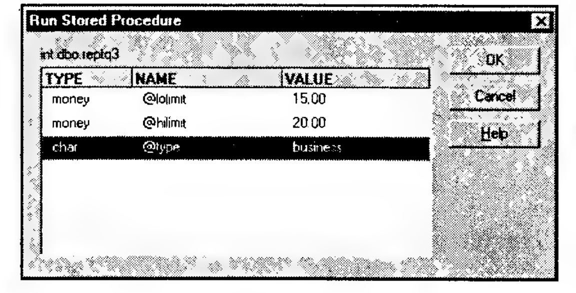


Рис. 23.16. Значения параметров для хранимой процедуры настраиваются в колонке Value окна Run Stored Procedure

Было бы неплохо использовать для передачи параметра type раскрывающийся список, содержащий все имеющиеся на данный момент в таблице значения соответствующего поля. Подобные возможности можно реализовать посредством создания программы на C++, использующей

данную хранимую процедуру SQL. Чтобы продемонстрировать на практике, как это делается, в следующем разделе мы подготовим новую хранимую процедуру и организуем ее вызов из программы на C++.

Подготовка новой хранимой процедуры

Для создания новой хранимой процедуры щелкните правой кнопкой мыши в окне DataView на ветви Stored Procedures и выберите в раскрывшемся контекстном меню команду New Stored Procedure. Раскроется окно редактора, содержащее следующий текст:

```
CREATE PROCEDURE /*Procedure_Name*/
AS
        Return (0)
```

Отредактируйте его так, как показано в листинге 23.1. Сохраните новую хранимую процедуру, и ее имя появится в окне DataView.

Листинг 23.1. Текст новой хранимой процедуры — author_ytd

```
CREATE PROCEDURE author_ytd @sales int
AS
SELECT authors.au_lname, authors.au_fname, titles.title, ytd_sales
        FROM authors, titles, titleauthor
        WHERE ytd_sales > @sales
                AND authors.au_id = titleauthor.au_id
                AND titleauthor.title_id = titles.title_id
ORDER BY ytd_sales DESC
```

Эта команда на языке SQL выбирает информацию из трех таблиц, используя столбцы au_id и title_id для соединения таблицы authors с таблицей titles. Процедуре передается один параметр sales, который принимает целочисленные значения. Запустите процедуру на выполнение, и результат ее работы можно будет увидеть в окне Output. В листинге 23.2 приведен результат, полученный при значении параметра sales, равном 4000.

Листинг 23.2. Результат работы процедуры author_ytd при @sales=4000

```
Running Stored Procedure dbo.author_ytd (@sales = 4000).

au_lname          au_fname    title                                        ytd_sales
----------- ------------ ---------------------------- --------

DeFrance          Michel      The Gourmet Microwave                         22246

Ringer            Anne        The Gourmet Microwave                         22246

Green             Marjorie    You Can Combat Computer Stress!               18722

Blotchet-Halls    Reginald    Fifty Years in Buckingham Palace Kitchens    15096

Carson            Cheryl      But Is It User Friendly?                      8780

Green             Marjorie    The Busy Executives Database Guide            4095

Bennet            Abraham     The Busy Executives Database Guide            4095

Straight          Dean        Straight Talk About Computers                 4095
```

```
Dull            Ann        Secrets of Silicon Valley                      4095

Hunter          Sheryl     Secrets of Silicon Valley                      4095

OLeary          Michael    Sushi, Anyone?                                 4095

Gringlesby      Burt       Sushi, Anyone?                                 4095

Yokomoto        Akiko      Sushi, Anyone?                                 4095

White           Johnson    Prolonged Data Deprivation: Four Case          4072
                           Studies

 (14 row(s) affected)
Finished running dbo.author_ytd.
RETURN_VALUE = 0
```

Подключение хранимой процедуры к программе на C++

К настоящему моменту у нас уже имеется заготовка приложения на C++, работающего с буфером выбранных данных и способного отобразить имеющиеся в нем записи (если в диалоговое окно добавить элементы управления для отображения значений столбцов). Определенная в приложении выборка данных содержит все столбцы из трех таблиц (authors, titleauthor и titles), заданных в процессе настройки AppWizard. Формирование выборки осуществляется функцией CPublishingSet::GetDefaultSQL(), которую AppWizard подготовил для созданного приложения. Текст этой функции представлен в листинге 23.3.

Листинг 23.3. Функция CPublishingSet::GetDefaultSQL(), сгенерированная AppWizard

```
CString CPublishingSet::GetDefaultSQL()
{
        return _T("[dbo].[authors],[dbo].[titleauthor],[dbo].[titles]");
}
```

Необходимо так изменить сгенерированное AppWizard выражение SQL, чтобы выполнялся вызов подготовленной нами хранимой процедуры, ставшей сейчас частью базы данных pubs. Прежде всего выберите команду Project⇨Set Active Project и выделите проект Publishing. В левой части экрана перейдите на вкладку ClassView, разверните содержимое класса CPublishingSet и сделайте двойной щелчок на функции GetDefaultSQL(). Раскроется окно редактора, содержащее текст этой функции. Приведите текст функции в соответствие с листингом 23.4.

Листинг 23.4. Функция CPublishingSet::GetDefaultSQL(), вызывающая хранимую процедуру

```
CString CPublishingSet::GetDefaultSQL()
{
        return _T("{CALL author_ytd(4000)}");
}
```

На заметку Как правило, не следует жестко кодировать в операторе вызова хранимой процедуры значение передаваемых ей параметров, как это выполнено в данном примере. О том, как добавить к классу переменные-члены, предназначенные для хранения значений параметров, и как передать их хранимым процедурам SQL, можно будет узнать из электронной документации позднее, когда вы приобретете определенный опыт работы с Enterprise Edition.

Записи, возвращаемые по запросам, помещаются в буфер выборки данных. По данному запросу возвращаются значения из четырех столбцов (au_lname, au_fname, title и ytd_sales), однако сейчас буфер выборки данных настроен на прием гораздо большего объема данных. С помощью мастера ClassWizard измените существующее определение выборки данных. С этой целью выполните следующие действия.

1. Раскройте на экране окно ClassWizard, выбрав команду View⇨ClassWizard.
2. Щелкните на корешке вкладки Member Variables. На экране вы должны увидеть все переменные-члены класса буфера выборки данных, связанные со столбцами таблиц (рис. 23.17).

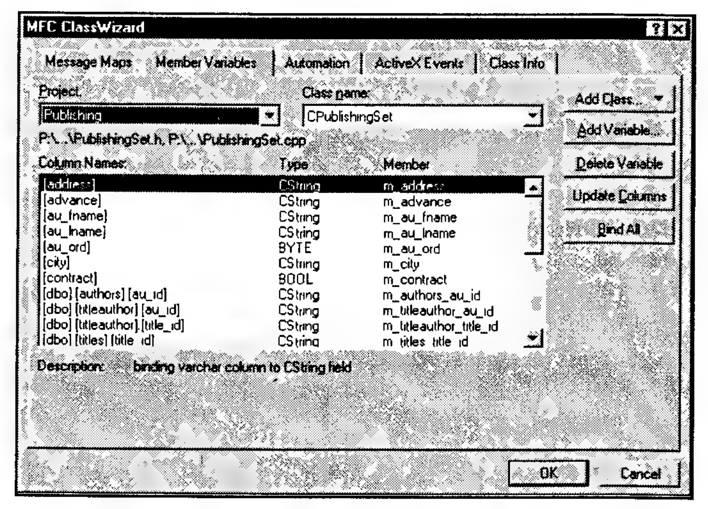


Рис. 23.17. С помощью ClassWizard можно корректировать определение класса выборки данных

3. Выделите имя колонки [address] и щелкните на кнопке Delete Variable.
4. Аналогичным образом удалите все остальные переменные, оставив только переменные au_lname, au_fname, title и ytd_sales.
5. Закройте окно ClassWizard, щелкнув на кнопке OK.

Теперь можно было бы откомпилировать создаваемое приложение и запустить его на выполнение, но пока не будут внесены изменения в диалоговое окно представления записей, вы не сможете увидеть записи и столбцы, возвращаемые каким бы то ни было запросом. О внесении изменений в определение диалоговых окон уже рассказывалось в главе 22, причем эти объяснения базировалось на сведениях, изложенных в главе 2. Поэтому приведенное здесь описание будет весьма кратким.

Щелкните на корешке вкладки ResourceView и в дереве ресурсов приложения разверните ветвь Dialogs, после чего сделайте двойной щелчок на идентификаторе IDD_PUBLISHING_FORM. Определение этого диалогового окна было автоматически создано мастером AppWizard, но в нем пока нет никаких элементов редактирования. Удалите из окна статический текст с напоминанием о необходимости добавить элементы управления, а затем добавьте в него четыре текстовых

поля редактирования и элементы статического текста для их названий (рис. 23.18). Вместо значений, предлагаемых Visual Studio по умолчанию, присвойте элементам редактирования осмысленные идентификаторы. Назовите их так: IDC_QUERY_LNAME, IDC_QUERY_FNAME, IDC_QUERY_TITLE и IDC_QUERY_YTDSALES.

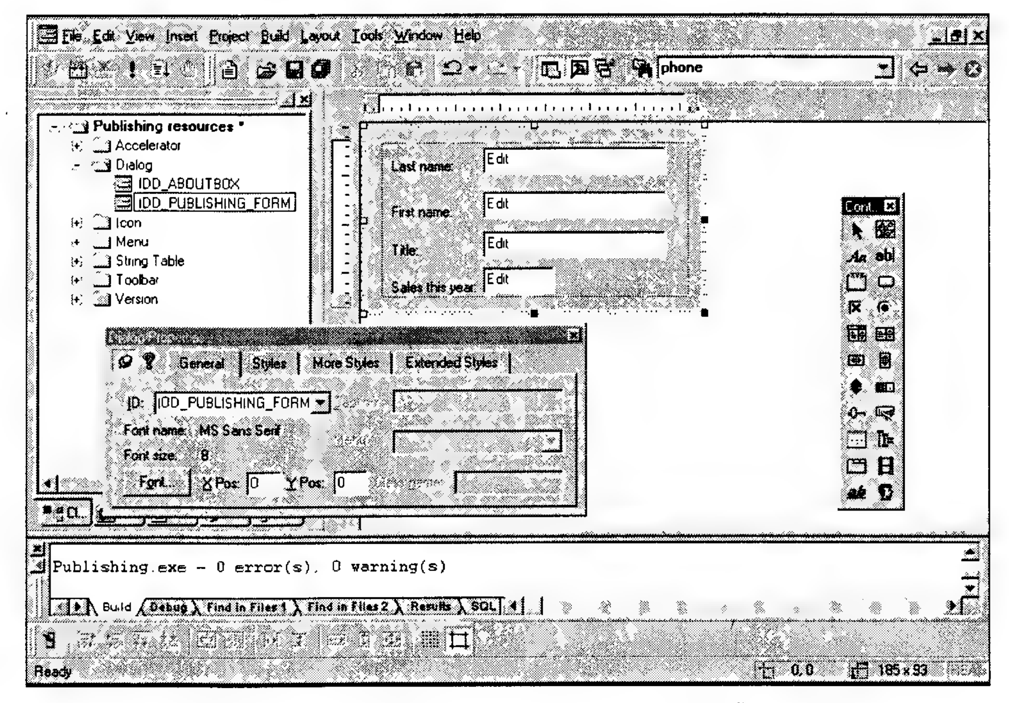


Рис. 23.18. Редактирование диалогового окна представления записей

Осталось последнее — связать эти элементы редактирования с переменными-членами. Для этого выполните следующее.

1. Пока редактируемое диалоговое окно находится в фокусе, раскройте окно мастера ClassWizard.
2. Щелкните на корешке вкладки Member Variables.
3. Выделите значение IDC_QUERY_FNAME и щелкните на кнопке Add Variable. Раскроется диалоговое окно Add Member Variable.
4. В раскрывающемся списке Member Variable name выделите значение m_pSet->m_au_fname и щелкните на кнопке OK.
5. Аналогичным образом свяжите переменную IDC_QUERY_LNAME с переменной m_pSet->m_au_lname, переменную IDC_QUERY_TITLE с переменной m_pSet->m_title и переменную IDC_QUERY_YTDSALES с переменной m_pSet->m_ytd_sales.
6. На рис. 23.19 показано диалоговое окно ClassWizard, в котором уже выполнено связывание для всех четырех элементов редактирования. Закройте окно, щелкнув на кнопке OK.

В окне ClassView щелкните дважды на имени функции DoFieldExchange() в ветви класса CPublishingSet и просмотрите сгенерированный код этой функции. Порядок, в котором в этом коде перечислены переменные, имеет существенное значение. Он должен соответствовать порядку перечисления возвращаемых полей в тексте хранимой процедуры, поэтому при необходимости откорректируйте текст функции.

Рис. 23.19. Связывание элементов редактирования окна представления с переменными-членами класса выборки данных

Оттранслируйте приложение и запустите его на выполнение. Вы должны увидеть на экране окно представления, подобное тому, которое приведено на рис. 23.20 (возможно, предварительно вам еще раз придется выполнить процедуру подключения к базе данных SQL). Используя клавиши управления курсором для переключения между выбранными записями, можно будет увидеть в окне представления данные обо всех авторах, перечисленных в отчете, который приведен в листинге 23.2.



Рис. 23.20. В окне приложения отображены результаты выполнения запроса, помещенного в хранимую процедуру



Прежде чем выполнять трансляцию приложения, убедитесь, что вы записали на диск подготовленную хранимую процедуру SQL. Поскольку хранимая процедура представляет собой отдельный субпроект основного проекта Publishing, запуск трансляции основного проекта не вызовет автоматического сохранения каких-либо элементов субпроекта.

На данный момент приложение способно выполнить немногое — оно просто вызывает хранимую процедуру и аккуратно отображает полученные результаты. Приложив немного усилий, вы, вероятно, сможете обеспечить из программы на C++ дружественный пользовательский интерфейс для хранимых процедур SQL. Вы непременно почувствуете простоту и удобство создания, отладки и поддержки хранимых процедур SQL с помощью средств Visual Studio.

Работа с базой данных

Окно DataView предоставляет полный набор средств контроля не только содержимого базы данных SQL, но и ее структуры. Комплекс графических инструментов позволяет наблюдать за работой базы данных и вносить любые изменения в ее функционирование.

`Database Designer` — конструктор баз данных

Вернитесь в окно DataView и сделайте щелчок правой кнопкой мыши на таблице `authors`, после чего выберите в раскрывшемся контекстном меню команду Design. Таким образом вы запустите `Database Designer`, который откроет диалоговое окно, показанное на рис. 23.21. В нем можно изменять назначения ключевых столбцов в таблице, устанавливать ширину столбцов, определять для столбцов допустимые значения и еще многое другое.

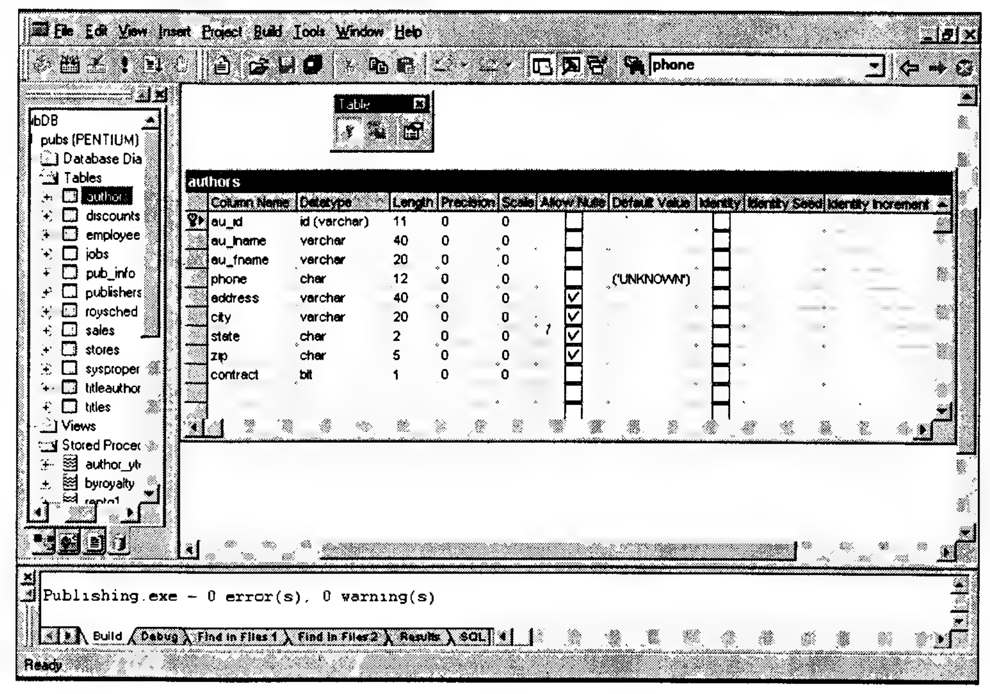


Рис. 23.21. `Database Designer` позволяет редактировать любые параметры структуры базы данных

К примеру, щелчок на пиктограмме Properties, расположенной справа на панели инструментов, в тот момент, когда выбран столбец au_id, приведет к выводу на экран окна свойств, показанного на рис. 23.22. Установленные в нем ограничения требуют, чтобы значение в столбце au_id было девятизначным числом. После щелчка на корешке вкладки Relationship эта вкладка диалогового окна Properties будет выведена на экран (рис. 23.23). Ее содержимое указывает на то, что столбец au_id используется для связывания таблицы authors с таблицей titleauthors.

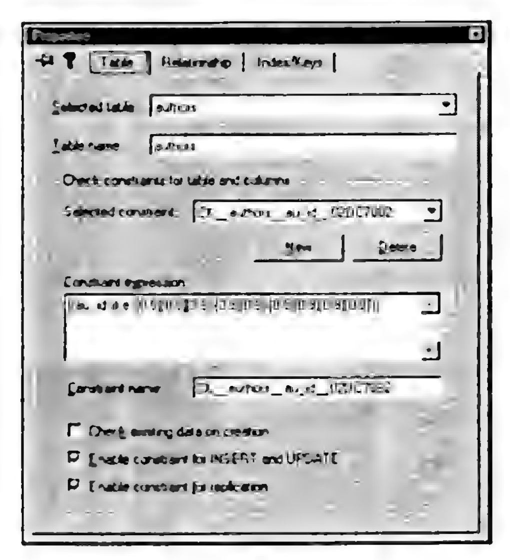


Рис. 23.22. Установка ограничений для значений столбца не составляет особого труда

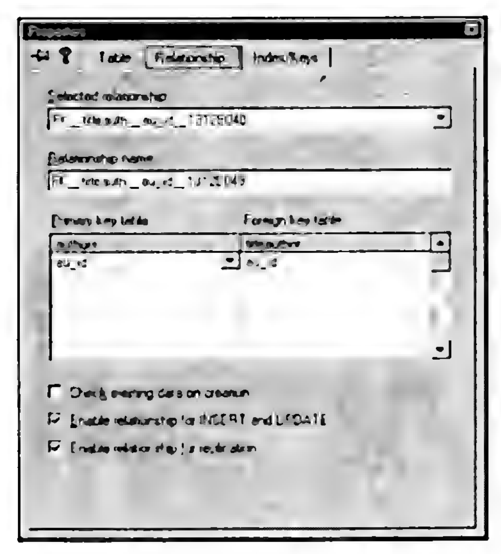


Рис. 23.23. Вкладка Relationship *упрощает анализ связей, установленных между таблицами*

Database Diagrams — конструктор диаграмм базы данных

Одним из самых простых и удобных способов предоставления информации о структуре внутренних связей в базе данных является диаграмма. На рис. 23.24 представлена диаграмма, которая описывает отношения между тремя таблицами, используемыми в примерах на протяжении всей этой главы.

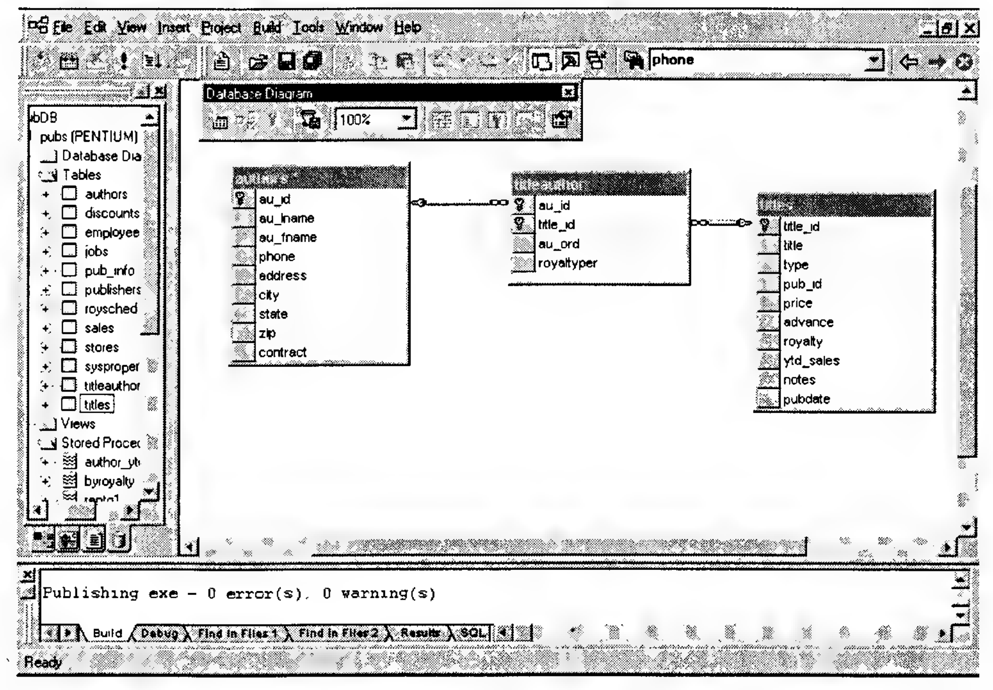


Рис. 23.24. Схематическое представление структуры базы данных

Чтобы самостоятельно создать подобную диаграмму, выполните следующие действия.

1. На панели DataView щелкните правой кнопкой мыши на элементе `Database Diagrams` и выберите в раскрывшемся контекстном меню команду New Diagram (Новая диаграмма).
2. Щелкните на таблице `authors` и перетащите ее в рабочую область.
3. Щелкните на таблице `titlesauthor` и перетащите ее в рабочую область. Подождите некоторое время, пока не появится связь между таблицами `authors` и `titlesauthor`.
4. Щелкните на таблице `titles` и перетащите ее в рабочую область. Опять подождите, пока не появится новая связь.
5. Расположите таблицы так, чтобы их ключи были на одной линии (см. рис. 23.24).
6. Перетащите указатели связи (стрелки) так, чтобы они соединяли соответствующие ключевые столбцы таблиц, как показано на рис. 23.24.

При желании, вы можете сохранить эту диаграмму в базе данных. Для этого достаточно щелкнуть на пиктограмме Save панели инструментов Standard и указать имя, под которым диаграмма будет сохранена. Созданной диаграммой смогут впоследствии пользоваться и другие разработчики, применяющие `Enterprise Edition` для доступа к этой базе данных.

Если вы являетесь разработчиком баз данных, то вам, наверное, уже не терпится открыть свою базу данных в окне Database Designer и приступить к работе. Не забывайте о возможностях, предоставляемых панелью инструментов Database Diagram. Например, пиктограмма New Text Annotation позволяет добавить аннотацию или описание. Четыре сгруппированных пиктограммы управляют степенью детализации при отображении каждой таблицы. Первая из них, Column Properties (Свойства столбца), выводит подробные сведения о каждой таблице. Вторая, Column Names (Названия столбцов), является выбранной по умолчанию при отображении таблиц. Пиктограмма Keys (Ключи) задает режим, при котором в окне будут отображены только те столбцы таблицы, которые являются ключевыми. Выбор пиктограммы Name Only (Только название) сожмет изображение и выведет лишь имя данной таблицы.

Для внесения каких-либо изменений в структуру данной таблицы выведите на экран контекстное меню и выберите в нем команду Column Properties, затем отредактируйте нужные свойства, аналогично тому, как вы это делали в окне Database Designer. Пожалуй, это действительно самый простой способ создания и сопровождения базы данных SQL. Не так ли?

Что такое Microsoft Transaction Server

Microsoft Transaction Server (MTS) — это отдельный программный продукт, поставляемый в составе пакета Visual C++ редакции Enterprise Edition, но не интегрированный с ним. MTS предоставляет возможность использовать набор объектов COM, называемых компонентами, для обеспечения надежного и безопасного выполнения распределенных транзакций внутри приложений, работающих с базой данных масштаба предприятия. Приложения, использующие MTS, могут быть написаны на любом языке, поддерживающем технологию ActiveX, — Visual C++, Visual J++ или Visual Basic.

Для работы с MTS вам потребуется знание основ программирования с использованием технологий ActiveX и COM, базирующихся на непосредственной работе с интерфейсами. Если вы всегда полагались на MFC, автоматически обеспечивающий всю работу с интерфейсами и скрывающий от вас все тонкости этого механизма, вам, вероятно, следует предварительно обратиться к главе 21, чтобы получить представление о том, что такое интерфейсы и как они используются.

MTS может использоваться практически с любыми видами баз данных, включая обыкновенные файловые системы, аналогично тому, как ODBC может быть использован почти со всеми СУБД. Безусловно, базы данных SQL будут работать с MTS, равно как и с огромным количеством других существующих менеджеров ресурсов. Следовательно, у вас есть возможность использовать всю мощь MTS при работе с ранее созданными СУБД.

Каждый компонент MTS является объектом COM. Он способен выполнять в системе некоторое конкретное задание. Часто несколько компонентов объединяются вместе, чтобы выполнить определенную *транзакцию*. Собранные вместе компоненты объединяются в пакеты (packages), подсоединяемые к системе в статусе модулей.

Транзакция — это последовательность действий, которая должна быть либо выполнена полностью, либо не выполнена совсем. Например, если покупатель переводит деньги с одного счета в банке на другой, деньги должны быть сняты с одного счета и положены на другой. Если один из этапов этого процесса будет выполнен, а другой нет, то полученные результаты не будут иметь смысла, поскольку либо у покупателя деньги будут несправедливо отняты, либо он приобретет их нечестным путем. Программисты, работающие с базами данных, давно выявили возможность подобных ошибок и разработали способы отката транзакций в случае, если произошел сбой на одном из промежуточных этапов. Кроме того, были разработаны способы предварительной проверки условий успешного завершения всех шагов транзакции

еще до начала ее выполнения. Однако эти технологии защиты очень сложно реализовать в больших распределенных системах — слишком сложно, чтобы выполнять это вручную.

К примеру, представим две системы, в которых должны быть выполнены операции снятия денег (скажем, по $100) с банковского счета покупателя. Прежде всего обе системы проверяют, имеется ли на счету клиента достаточная сумма. Обе системы подключены через сеть к базе данных, в которой хранится текущий остаток на данном счете. Первая система запрашивает размер текущего остатка и получает ответ — $150. Мгновением позже вторая система запрашивает размер остатка, и ей сообщается о тех же $150. Первая система уверенно посылает запрос на перевод $100 и успешно их получает. Вторая система посылает запрос на перевод $100 на долю секунды позже и не получает этой суммы. Во второй системе все выполненные шаги транзакции, связанной с данным клиентом, должны быть отменены. Системы обработки транзакций, подобные MTS, значительно упрощают реализацию подобных процессов, предоставляя системные службы, которые обеспечивают решение этих задач.

Ну что, произвело впечатление? Тогда устанавливайте данный продукт и изучайте предоставленную в комплекте с ним электронную документацию. В нее включены два превосходных примера реальных систем — простое приложение для банка и игра. Вы также можете обратиться за информацией на страницу Web для Microsoft's Transaction Server по адресу **http://www.microsoft.com/transaction**.

Использование Visual SourceSafe

Если вы являетесь членом группы разработчиков, то система отслеживания изменений (revision control system) будет для вас не просто полезной, а жизненно необходимой. Для абсолютного большинства таких групп функции системы отслеживания изменений заключаются в оповещении ваших коллег о том, что вы собираетесь некоторое время поработать, например, над файлами `fooble.h` и `fooble.cpp`, сопровождая это просьбой пока не вносить в них никаких изменений. Весьма вероятно, что вам, кроме того, потребуется выяснить, кто сохранил внесенные в `fooble.h` изменения поверх изменений, сделанных вами. Поскольку вы одновременно открыли файл, этот кто-то сохранил сделанные им изменения после того, как это сделали вы. Подобная практика не сулит ничего хорошего, однако выход есть.

Системы отслеживания изменений отнюдь не новы. Все они тем или иным образом реализуют следующие функции.

- **Захват файла** (check out a file). Выгружая копию файла из архива или центральной библиотеки на свой рабочий компьютер, вы одновременно помечаете данный файл как недоступный для всех, кто захочет внести в него изменения. (Некоторые системы позволяют вносить изменения в исходный файл одновременно нескольким разработчиками, а потом осуществляют слияние внесенных изменений.)
- **Освобождение файла** (check in a file). После окончания внесения изменений вы возвращаете файл в библиотеку. Нужно предоставить краткое описание выполненных модификаций, а система отслеживания изменений автоматически добавит к описанию ваше имя, дату и отметит другие файлы, на которые распространяются внесенные изменения.
- **Слияние изменений** (merge changes). Некоторые системы отслеживания изменений могут принимать небольшие изменения, одновременно внесенные разными разработчиками в один и тот же файл, и гарантируют, что все выполненные изменения будут зафиксированы в итоговом файле.
- **Регистрация изменений** (change tracking). Некоторые системы отслеживания изменений могут восстанавливать предыдущие версии файла, обрабатывая в обратном порядке информацию, которая зафиксирована в журнале регистрации изменений.

- **Предыстория** (history). Информация о проведенных изменениях, содержащаяся в справках, может послужить основой для составления сводок изменений, внесенных в каждый из файлов, с указанием, когда и зачем они были выполнены.

Приложение Microsoft Visual SourceSafe является отличной системой отслеживания изменений, которую многие разработчики используют с целью обеспечить порядок в своих архивах исходных текстов программ. Что же выделяет Visual SourceSafe среди других подобных систем? Ее ориентированность на проекты и глубокая интеграция с Visual C++ (с помощью нового интерфейса SCCI, который используется в некоторых подобных системах). Кроме того, она поставляется в составе редакции Enterprise Edition пакета Visual C++.

Для инсталляции приложения Visual SourceSafe укажите режим выборочной (custom) установки и выберите опцию Enable SourceSafe Integration. В результате в меню Project среды Visual Studio будет помещено подменю, показанное на рис. 23.25. Для того чтобы эти пункты в меню были разблокированы, необходимо подключить проект к системе отслеживания изменений, выбрав команду Add to Source Control, и зарегистрироваться в Visual SourceSafe.

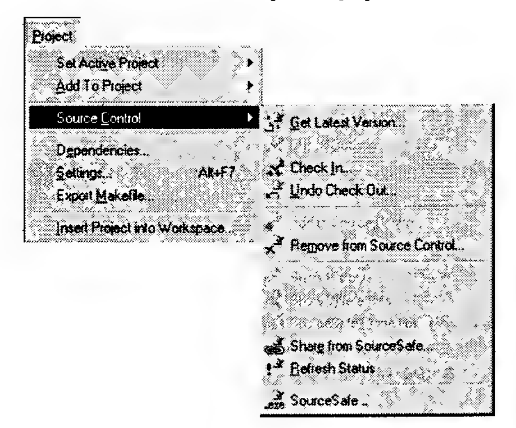


Рис. 23.25. После инсталляции Visual SourceSafe в меню Project будет добавлено новое подменю

Данное подменю содержит следующие команды.

- Get Latest Version (Получить последние изменения) — заменяет существующие у вас копии выделенных файлов более новыми копиями из библиотеки.
- Check Out (Захват файла) — начать работу с файлом.
- Check In (Освобождение файла) — заканчивает работу с файлом и делает измененную вами версию доступной всем.
- Undo Check Out (Отменить захват файла) — возвращает файл без фиксации изменений и внесения извещения в архив.
- Add To Source Control (Добавить к системе контроля) — подключает данный проект к системе отслеживания изменений.
- Remove From Source Control (Исключить из системы контроля) — исключает данный проект из системы отслеживания изменений.
- Show History (Показать архив) — отображает изменения, сделанные в выделенных файлах.

- **Show Differences** (Показать отличия) — отображает отличия между старой и новой версиями файлов.
- **SourceSafe Properties** (Свойства SourceSafe) — отображает информацию о ваших файлах, хранимую в SourceSafe.
- **Share From SourceSafe** (Совместное использование в SourceSafe) — позволяет другим разработчикам работать с выделенными файлами.
- **Refresh Status** (Обновить статус) — обновляет отображаемые у вас данные с учетом изменений, внесенных другими разработчиками.
- **Source Safe** (Запустить SourceSafe) — запускает Visual SourceSafe для отображения отчетов и сводок.

Прежде чем вам будет предоставлена возможность подключения проекта к системе отслеживания изменений и обеспечено использование указанных выше возможностей, вам должен быть присвоен личный идентификатор и пароль для входа в систему Visual SourceSafe. Запустите с помощью команд описанного выше меню программу Visual SourceSafe — и она решит любые административные задачи, которые отныне можно будет вычеркнуть из списка ваших собственных обязанностей.

Если вы не являетесь единственным разработчиком, выполняющим данный проект, вы просто обязаны использовать систему отслеживания изменений. Visual SourceSafe является прекрасной системой, которая работает непосредственно в среде Visual Studio. Если вы уже приобрели редакцию Enterprise Edition пакета Visual C++, она достанется вам бесплатно. Что еще остается желать? Инсталлируйте ее, изучите и пользуйтесь. Вы не пожалеете!



Система отслеживания изменений одинаково хорошо работает как с файлами программ, так и со страницами Web, файлами баз данных, документацией, списками ошибок и файлами электронных таблиц. Как только работа с ней войдет у вас в привычку и вы ощутите реальные преимущества ее использования, вам не захочется останавливаться на достигнутом.

ГЛАВА 24

Повышение производительности приложений

В этой главе...

Макросы ASSERT и TRACE

Отладочные функции

Устранение утечки памяти

Оптимизация

Профилирование

Создавая новое приложение, разработчики вынуждены преодолевать множество препятствий. Нужно добиться, чтобы приложение нормально откомпилировалось и бесперебойно работало, причем функционировало именно так, как изначально задумано разработчиком. При планировании некоторых проектов специально резервируется время в графике работ на выяснение того, можно ли заставить приложение работать быстрее, использовать меньше памяти или же обеспечить меньший размер выполняемого файла. Обсуждаемые в этой главе технологии повышения производительности и совершенствования эксплуатационных характеристик создаваемых приложений помогут обеспечить надежность программ и устранить имеющиеся в них погрешности, приводящие к неверным результатам вычислений или ошибкам в отчетах. Очень часто такие изменения могут оказаться чем-то гораздо более существенным, чем просто нанесение последнего штриха и наведение глянца на готовом продукте.

У вас должно войти в привычку добавлять в текст программы *элементы защиты* непосредственно во время ее написания и использовать предоставляемые Visual Studio инструменты отладки с тем, чтобы иметь полное представление о реальном состоянии дел во время выполнения программы. Если вы отложите тестирование программы на самый конец, то и тестирование, и исправление ошибок выполнить будет намного труднее, нежели осуществить поэтапную отладку фрагментов программы в процессе ее создания. Ну и, конечно же, все заранее обнаруженные и устраненные ошибки уже не потребуется исправлять в дальнейшем!

Макросы ASSERT и TRACE

Концепции контроля промежуточных данных и трассировки не были изобретены создателями пакета Visual C++. Эти идеи воплощены и в других системах программирования, они включены в тематику многочисленных учебных программ и курсов по программированию. Преимущества реализации этих принципов в Visual C++ заключаются в четкой организации представления результатов и легкости, с которой операторы контроля и трассировки могут быть удалены из окончательной версии приложения.

Макрос ASSERT: обнаружение логических ошибок

Макрос ASSERT позволяет проверить текущее состояние логических операторов и "забить тревогу" в случае, если это состояние не есть TRUE (истина). К примеру, предположим, что вы собираетесь обратиться к элементу массива следующим образом:

```
array[i] = 5;
```

Этот оператор выполнится нормально, если индекс i не принимает значение, меньшее нуля, и не превосходит число распределенных для массива элементов. Это должен обеспечить предшествующий программный код вычисления i, если в нем нет ошибок, но не грех бы удостовериться в правильности результата. Приведенный ниже оператор ASSERT будет проверять выполнение логического условия, заключенного в круглые скобки:

```
ASSERT( i > 0 && i < ARRAYSIZE)
```

На заметку В конце данной строки отсутствует точка с запятой (;), так как ASSERT является *макросом*, а не функцией. Написанные ранее программы на языке С могли вызывать функцию с именем assert(), однако мы настоятельно рекомендуем заменить такие вызовы макросом ASSERT, так как эти макросы удаляются во время окончательной "сборки" программы, о чем еще будет идти речь ниже в этой главе.

Макросы ASSERT предназначены для проверки логики работы программы. Они ни в коем случае не должны использоваться для проверки допустимости введенных пользователем или хранящихся в файлах данных. Всякий раз, когда результат вычисления условного выражения

внутри оператора ASSERT будет иметь значение FALSE, выполнение программы будет остановлено и появится сообщение о том, какой именно макрос контроля стал этому причиной. В итоге можно будет точно узнать, в какой именно точке программы есть логическая или конструктивная ошибка, которую необходимо исправить. Вот другой пример:

```
// Вызывающий код должен передавать непустой указатель.
void ProcessObject( Foo * fooObject )
{
        ASSERT( fooObject )
        // Обработка объекта.
}
```

В этом фрагменте программы можно безо всяких опасений использовать указатель, будучи уверенным, что выполнение программы будет остановлено, если значение переданного указателя — NULL.

Visual Studio позволяет создавать как отладочную, так и окончательную версии программы. В отладочной версии директива #define определяет константу с именем _DEBUG, в результате чего в макросах и прочем препроцессорном тексте программы по наличию этой константы можно распознать тип создаваемой версии. Если константа _DEBUG не определена, в макросах ASSERT не выполняется никакой обработки. Это означает, что в окончательной версии программы отсутствует потеря производительности, связанная с выполнением операторов if, контролирующих наличие логических ошибок. При этом не нужно пересматривать весь исходный текст для удаления конструкций ASSERT в готовой версии приложения. Больше того, лучше оставить их на месте для разработчиков, которым в дальнейшем придется модифицировать данную программу. Однако макросы ASSERT не смогут вам помочь при возникновении проблем в распространяемой версии приложения, поскольку они используется для выявления логических и конструктивных ошибок *до создания* окончательной версии программного продукта.

Макрос TRACE: определение места возникновения ошибки

Отладчик среды разработки Visual Studio обеспечивает выполнение всех мыслимых функций, необходимых при отладке приложения. Выполнение программы может осуществляться по шагам — один оператор исходного текста за шаг — или непрерывно, но с остановками в заданных точках. В процессе выполнения программы значения любых переменных можно увидеть в окне наблюдения (Watch Window). Однако подобные процедуры требуют больших затрат времени, поэтому многие разработчики используют макросы TRACE и с их помощью быстро обнаруживают критические точки в отлаживаемой программе. Локализовав такую точку, можно перейти к традиционной пошаговой отладке для выявления ошибки в тексте программы.

Прежняя технология поиска фрагментов кода, содержащих ошибку, предполагала включение в программу многочисленных операторов print, что представляет собой определенную проблему для Windows-приложений. Сейчас не спешите приступать к поиску возможных обходных путей, скажем, печати в файл, — макрос TRACE способен выполнить все, что вам нужно. И, как и ASSERT, он волшебным образом исчезнет при создании окончательной версии программы.

На самом деле существует несколько TRACE-макросов: TRACE, TRACE0, TRACE1, TRACE2 и TRACE3. Численный суффикс в имени указывает на число параметрических аргументов, следующих за обычной строковой константой, что во многом напоминает оператор printf. Различные версии TRACE включены в Visual C++ с целью экономии места в сегменте данных.

Когда приложение создается с помощью мастера AppWizard, в его текст автоматически добавляются многочисленные макросы ASSERT и TRACE. Вот пример использования макроса TRACE:

```
if (!m_wndToolBar.Create(this)
        || !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
{
        TRACE0("Failed to create toolbar\n");
        return -1; // Создать не удалось.
}
```

Если создание (или загрузка) панели инструментов завершилось неудачей, соответствующая подпрограмма возвратит значение –1, что для вызывающей программы послужит сигналом о наличии сбоя. Данная схема будет работать как в отладочной, так и в окончательной версиях программы. Однако в отладочной версии программы макрос TRACE пошлет на экран сообщение, которое поможет программисту обнаружить причину возникновения ошибки.

Все макросы TRACE посылают свои сообщения в поток afxDump, обычно представляющий собой окно отладки, но в случае приложений, работающих с консолью, их сообщения могут быть посланы и в поток stderr. Числовой суффикс в имени определяет количество параметрических аргументов, которые можно использовать в строковой константе для определения типа передаваемых данных. Например, для передачи оператором TRACE сообщения, которое содержит значение целочисленной переменной, он должен иметь такой вид:

```
TRACE1("Error Number: %d\n", -1 );
```

Для передачи двух аргументов, скажем, строковой константы и целого числа, оператор должен быть таким:

```
TRACE2("File Error %s, error number: %d\n", _FILE_, -1 );
```

Самое трудное в трассировке — это выработать в себе привычку пользоваться ею. Расставляйте конструкции TRACE везде, где в программе анализируются индикаторы ошибок, перед макросами ASSERT и в тех местах, где, как вам кажется, текст программы не совсем корректен. Столкнувшись с неожиданным поведением программы, прежде всего добавьте в нее конструкции TRACE, что позволит разобраться в происходящем еще до начала пошаговой отладки.

Отладочные функции

Если вам понравилась идея применения исходного текста, не включаемого в окончательную версию программы, возможно, у вас появится желание самостоятельно подготовить подобный текст, который будет включен в отладочную версию программы, но будет отсутствовать в окончательной. Это легко организовать. Достаточно окружить такой текст директивами проверки наличия константы _DEBUG, как показано ниже:

```
#ifdef _DEBUG
        // Поместите сюда отладочный текст.
#endif
```

При создании окончательной версии программы этот текст вообще не будет компилироваться.

Кроме того, при создании отладочной и окончательной версий программы можно использовать различные опции. Многие разработчики, например, используют для этих процедур различные уровни предупреждений компилятора. Все установки и опции конфигурирования компилятора и компоновщика для процедур создания отладочной и окончательной версий программы хранятся отдельно и могут быть изменены независимо друг от друга. К примеру, для установки 4-го уровня предупреждений транслятора только лишь при компиляции в отладочном режиме выполните следующие действия.

1. Выберите команду Project⇨Settings. Раскроется диалоговое окно Project Settings, показанное на рис. 24.1.

2. В раскрывающемся списке, расположенном слева в верхней части окна, выберите одно из значений — Debug (Отладка) или Release (Конечная версия). Если вы выберете значение All Configurations (Все конфигурации), то выполненные установки будут относиться одновременно и к отладочной, и к окончательной версиям.

3. Щелкните на корешке вкладки C/C++ и в раскрывающемся списке Warning Level выберите значение Level 4, как показано на рис. 24.2. По умолчанию принимается значение Level 3, которое будет использоваться при создании конечной версии (рис. 24.3).

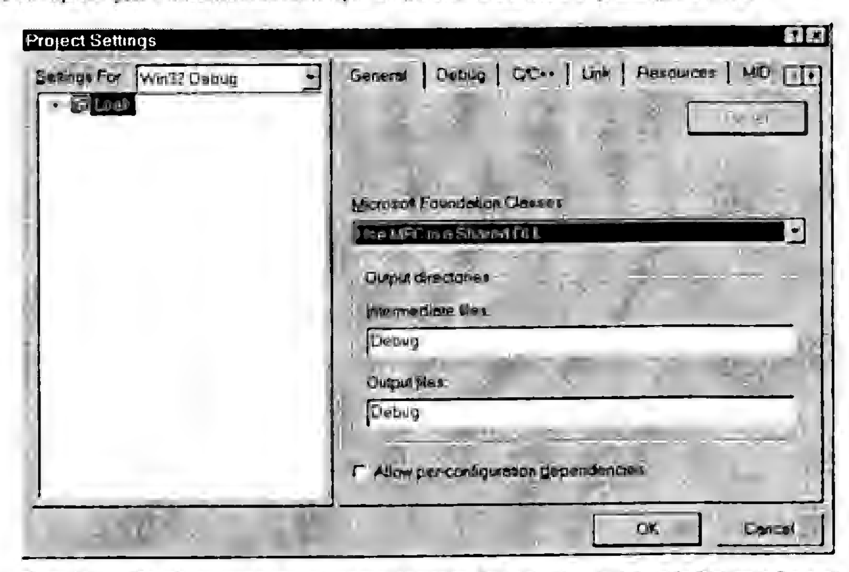


Рис. 24.1. Конфигурирование транслятора для различных стадий разработки выполняется в диалоговом окне Project Settings

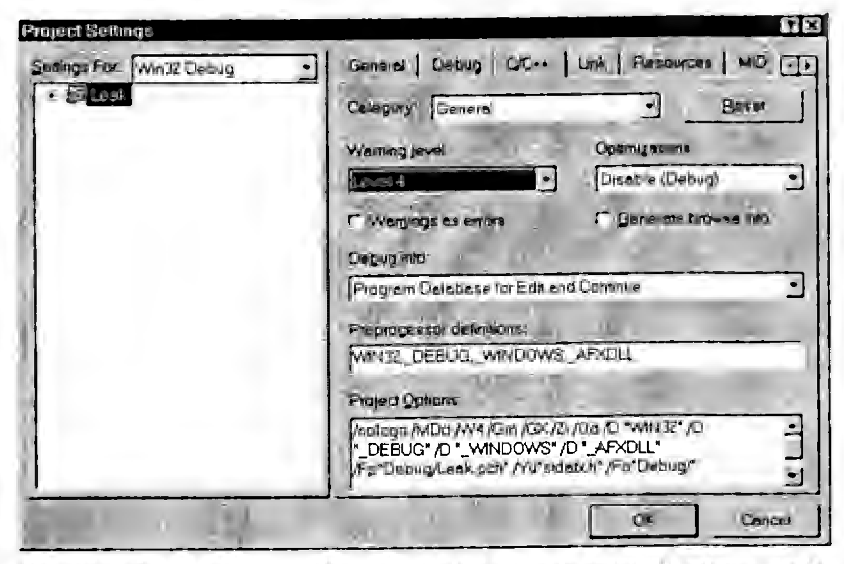


Рис. 24.2. В процессе разработки используется повышенный уровень предупреждений транслятора

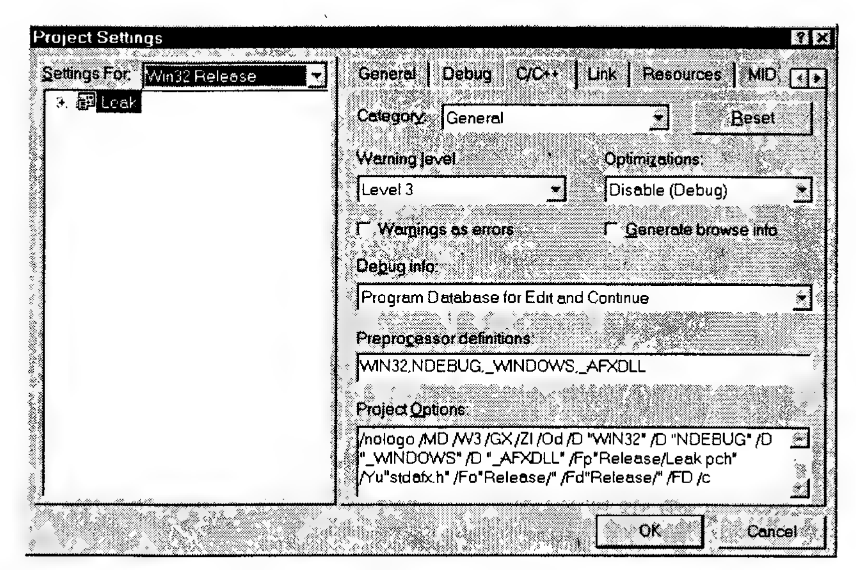


Рис. 24.3. При построении окончательной версии приложения обычно используется более низкий уровень предупреждений транслятора

Устранение утечки памяти

Утечку памяти (memory leaks) можно считать наиболее пагубной из всех ошибок. Небольшие утечки могут не вызывать осложнений в работе программы до тех пор, пока однажды она не проработает достаточно долгое время или пока в обработку не поступит необычно большой файл данных. Поскольку большинство программистов тестируют создаваемые приложения с помощью файлов данных небольшого размера и при исследовании поведения программы запускают ее всего на несколько минут, утечки памяти могут не обнаружиться при обычном тестировании. Однако они могут явно обнаружиться во время реальной эксплуатации программы пользователем и привести к аварийной остановке программы или возникновению других ошибок.

Основные причины возникновения утечки памяти

Что же представляет собой утечка памяти в программе? Это означает, что программа не освобождает память, выделенную ей по запросу. Первой и самой простой причиной является несбалансированность операторов выделения и освобождения памяти из кучи под объект или массив объектов — `new` и `delete`. Другой причиной возникновения утечки памяти является изменение значения указателя без освобождения памяти, адрес которой хранился в этом указателе. Много неявных утечек памяти возникает, когда класс, имеющий указатель в качестве переменной-члена, выполняет оператор `new` для присвоения значения указателю, но не имеет собственного конструктора копирования, терминального оператора присвоения или деструктора. В листинге 24.1 содержится текст, иллюстрирующий некоторые причины, приводящие к утечке памяти.

Листинг 24.1. Примеры, иллюстрирующие причины возникновения утечки памяти в программах

```
// Потеря указателя на область памяти.
{
        int * one = new int;
        *one = 1;
}       // Выделенная новая область памяти теперь недоступна, но
        // она не была удалена.

// Несоответствие new и delete:
// new должно быть сопряжено с delete,
// a new[] - с delete[].
{
        float * f = new float[10];
. . . // Использование этого массива.
        delete f;          // Ошибка! Будет удалено только f[0],
                           // следует использовать delete[] f;.
}

// Потеря значения указателя на выделенный участок памяти без ее высвобождения.
{
        const char * DeleteP = "Dont forget P";
        char * p = new char[strlen(DeleteP) + 1];
        strcpy (p, DeleteP);
}
// Указатель получил новое значение без выполнения delete[].

class A
{
        public:
                int * pi;
}
A::A()
{
        pi = new int();
        *pi = 3;
}
//... далее, этот класс используется в некотором фрагменте программы...
A firsta;          //Выделяется память под переменную типа int, которую
                   // адресует указатель firsta.pi.
A seconda;// Выделяется память под еще одну переменную типа int,
                   // которую адресует указатель seconda.pi.
seconda = firsta;
// Выполняется побитовое копирование. Оба объекта будут иметь
// указатель pi, адресующий первую выделенную переменную
// типа int. Адрес второй переменной типа int будет утрачен.
```

Все эти фрагменты исходного текста демонстрируют варианты ситуации, когда указатели на некоторую выделенную область памяти теряются до ее освобождения. Когда значение указателя на область памяти утрачивается, вы уже не можете использовать адресованную им память, а также никто другой не сможет использовать этот участок памяти. Дело может принять совсем плохой оборот, если вы вспомните о существовании исключений, обсуждаемых в главе 26. Проблема в том, что, если вызывается исключение, процесс выполнения может покинуть функцию еще до того, как будет выполнен оператор delete, расположенный в конце текста функции. Поскольку в процессе освобождения стека вызываются деструкторы объектов, выпавших из области видимости, существует возможность предотвратить подобные неприятности с помощью операторов delete в деструкторе. Об этом более детально рассказывается в главе 26.

Как и в случае возникновения любых других ошибок, секрет эффективной борьбы с утечками памяти заключается в том, чтобы предотвратить возможность их появления или, как минимум, обнаружить их появление на самых ранних этапах. Выработайте в себе несколько полезных привычек.

- Если класс содержит указатель и присваивает ему значение с помощью оператора new, обязательно создайте деструктор, который освобождает этот участок памяти. Кроме того, напишите конструктор для операции копирования и перегрузите терминальный оператор присвоения (=).
- Если функция запрашивает выделение памяти и возвращает нечто, предоставляющее вызывающей программе доступ к этой памяти, то она должна возвращать указатель, а не ссылку. Освободить память по ссылке невозможно.
- Если функция запрашивает выделение памяти, а позднее эта же функция ее освобождает, предпочтительнее размещать эту память в стеке, для того, чтобы вы не забыли ее освободить.
- Никогда не изменяйте значения указателя, пока не будет удален объект или массив, который им адресован. Никогда не используйте операцию приращения для указателя, значение которого установлено оператором new.

Отладочные версии операторов new и delete

MFC может предложить программисту, занятому поисками причин утечки памяти, много различных инструментов. При создании отладочной версии приложения везде, где вы используете операторы new и delete, на самом деле используются их специальные отладочные версии, отслеживающие название файла и номер строки, в которых выполняется каждое выделение памяти. Затем все операторы delete сопоставляются с сопряженными операторами new. Если по окончании работы программы будет обнаружен остаток памяти, не возвращенной системе, в окне вывода появится предупреждающее сообщение, показанное на рис. 24.4.

Для того чтобы увидеть это сообщение собственными глазами, создайте с помощью мастера AppWizard MDI-приложение с названием Leak, приняв при настройке мастера все предлагаемые по умолчанию установки. В функции InitInstance() класса приложения (для данного примера — CLeakApp) добавьте следующую строку:

```
int* pi = new int[20];
```

Оттранслируйте отладочную версию приложения и запустите ее на выполнение, выбрав команду Build⇨Start Debug⇨Go или щелкнув на пиктограмме Go панели инструментов Build. Результаты выполнения программы должны быть приблизительно такими, как на рис. 24.4. Обратите внимание, что название файла (Leak.cpp) и номер строки (54), в которой выделялась память, указаны непосредственно в сообщении об ошибке. В окне редактора будет отображен файл Leak.cpp с курсором, помещенным на строку 54. (Координаты в правом углу строки состояния окна всегда напоминают о местоположении курсора.) Если бы вы работали над настоящим приложением, то представленная информация точно бы указала причину возникшей проблемы. Далее нужно только решить, как с ней совладать (точнее, решить, где следует разместить соответствующий оператор delete).

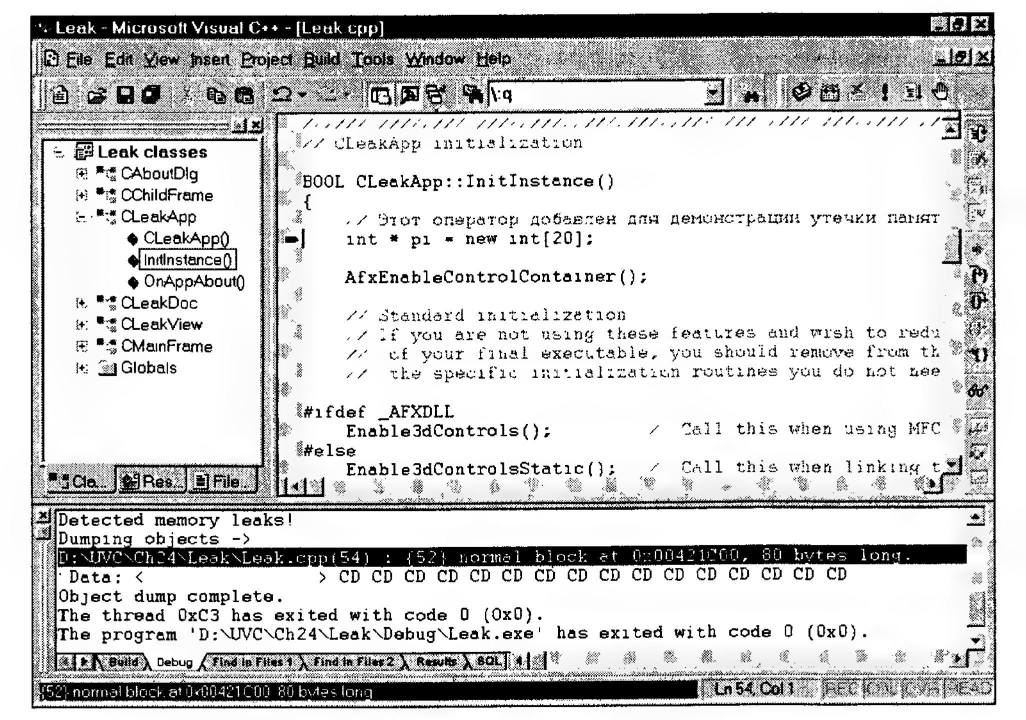


Рис. 24.4. В отладочной версии приложения утечки памяти обнаруживаются автоматически

Автоматические указатели

Когда программа передает управление некоторому своему блоку, например функции, память для всех локальных переменных этой функции выделяется в стеке. *Стек* — это временное хранилище, размеры которого могут уменьшаться или увеличиваться. Стек используется для запоминания точки входа в функцию, аргументов, передаваемых функции, а также локальных объектов и переменных функции.

Когда функция возвращает управление, *указатель стека* возвращается на точку входа в функцию. При этом элементы, помещенные туда функцией, считаются уничтоженными, что делает возможным дальнейшее использование этой памяти в других процессах. Данная процедура называется *освобождением стека.*

На заметку Объекты и переменные, определенные с ключевым словом `static`, размещаются не в стеке.

Кроме того, освобождение стека выполняется и при вызове исключения. Для надежного восстановления того состояния программы, в котором она пребывала до вызова исключения, выполняется процесс освобождения стека. Переменные, размещенные в стеке, уничтожаются, а для размещенных в стеке объектов вызываются деструкторы, уничтожающие их. К сожалению, с динамическими объектами дело обстоит иначе. Дескрипторы (например, указатели) также уничтожаются при освобождении стека, однако оператор `delete` при этом не вызывается. В результате возникает утечка памяти.

В некоторых случаях для решения проблемы достаточно добавить операторы `delete` в деструкторы тех объектов, о которых точно известно, что они будут уничтожены в процессе ос-

вобождения стека. Эти деструкторы смогут использовать значения указателей до того, как последние выйдут из области видимости. Наиболее общим подходом является замена простых указателей классом C++. Объект такого класса можно использовать просто как указатель, но при этом он содержит деструктор, который освобождает адресуемую таким указателем память. Не беспокойтесь — вам не потребуется создавать такой класс, ведь он уже включен в библиотеку Standard Template Library, входящую в состав пакета Visual C++. В листинге 24.2 содержится упрощенная версия определения класса auto_ptr, предназначенная для иллюстрации основных принципов его функционирования.



Если вам еще не приходилось видеть текст шаблона, то обратитесь к главе 26.

Листинг 24.2. Упрощенная версия определения класса auto_ptr

```
// Это определение класса не является полным.
// Пользуйтесь полным определением в Standard Template Library.
template <class T>
class auto_ptr
{
public:
        auto_ptr( T *p = 0) : rep(p) {}
        // Указатель хранится в классе.
        ~auto_ptr(){ delete rep; } //Удаляет внутренний rep.
        // Включение членов для преобразования указателей.
        inline T* operator->() const { return rep; }
        inline T& operator* () const { return *rep; }
private:
        T * rep;
};
```

Этот класс имеет одну переменную-член — указатель на любой тип. Класс также имеет конструктор с одним аргументом для создания указателя auto_ptr из указателей типа int*, Truck* или любого другого. Деструктор освобождает память, которую адресует внутренняя переменная-член. Наконец, в классе перегружены операторы раскрытия ссылки -> и * для того, чтобы раскрытие ссылки auto_ptr было похоже на раскрытие ссылки простого указателя.

Оператор создания автоматического указателя с именем p на объект некоторого класса C будет выглядеть следующим образом:

```
auto_ptr<C> p(new C());
```

Теперь можно использовать указатель p так, как будто это просто указатель C*, например:

```
p->Method();     // Вызывает функцию C::Method().
```

Нет необходимости удалять объект класса C, на который указывает p, даже в случае исключения, поскольку адресуемый p-объект размещается в стеке. Когда он выходит из области видимости, вызывается его деструктор, в котором и выполняется оператор delete класса C для объекта, размещенного с помощью оператора new.

Оптимизация

Было время, когда программистам самим приходилось оптимизировать созданный ими исходный текст программы. Много ночей проходило в раздумьях о том, в каком порядке следует проверять условия или какие переменные следует хранить в регистрах, а не в оперативной памяти. Сегодня компиляторы поставляются в комплекте с оптимизаторами, способными ускорить выполнение программы или уменьшить ее размер.

Вот простой пример работы оптимизаторов. Представим, что вы написали следующий фрагмент текста программы:

```
for (i=0; i<10;i++)
{
        y=2;
        x[i]=5;
}
for (i=0; i<10; i++)
{
        total += x[i];
}
```

Этот код будет работать быстрее, если выражение `y=2` поместить перед первым циклом. К тому же эти два цикла легко могут быть объединены в один. Простое увеличение `total` на 5 будет выполнятся значительно быстрее, чем то же самое, но с предварительным вычислением адреса `x[i]` лишь для того, чтобы получить значение, только что туда же помещенное. Хорошие оптимизаторы могут даже "сообразить", что `total` с успехом можно подсчитать и вне цикла. Откорректированный текст программы может выглядеть следующим образом:

```
y=2;
for (i=0; i<10; i++)
{
        x[i]=5;
}
total += 50;
```

Конечно же, оптимизаторы способны на большее, но этот простой пример дает представление, что происходит там, за кулисами. Можно выбрать в качестве критерия оптимизации скорость, что, как правило, происходит за счет использования дополнительной памяти, или же потребовать минимизировать использование памяти, возможно, за счет определенного снижения скорости выполнения.

Для задания в проекте опций оптимизации выберите на панели меню Visual Studio команду Project⇨Settings. Раскроется окно свойств Project Settings, показанное впервые на рис. 24.1. Щелкните на корешке вкладки C/C++ и убедитесь, что вы работаете с опциями, используемыми при создании конечной версии проекта. Далее в раскрывающемся списке Category выберите значение `Optimizations` (Оптимизация). При создании отладочных вариантов программы оптимизация должна быть отключена, иначе текст программ в исходных файлах и выполняемый код не будут совпадать строка в строку, что будет сбивать с толку и вас, и отладчик. Определенный тип оптимизации следует устанавливать только в окончательной версии программы. Выберите любой существующий тип оптимизации из раскрывающегося списка Optimization, как показано на рис. 24.5.

Если в списке Optimizations выбрать опцию `Customize`, можно будет выбрать из дополнительного списка отдельные варианты методов оптимизации, включая `Global Optimizations` (глобальная оптимизация), `Favor Fast Code` (приоритет скорости выполнения), `Generate Intrinsic Functions` (создавать встроенные функции) и т.д. Однако, как видно из приведенных выше названий опций оптимизации, перед выбором схемы `Customize` следует тщательно продумать, что именно вы хотите получить от оптимизатора. А пока рекомендуется принять одну из готовых схем оптимизации, заранее подготовленных создателями Visual C++.

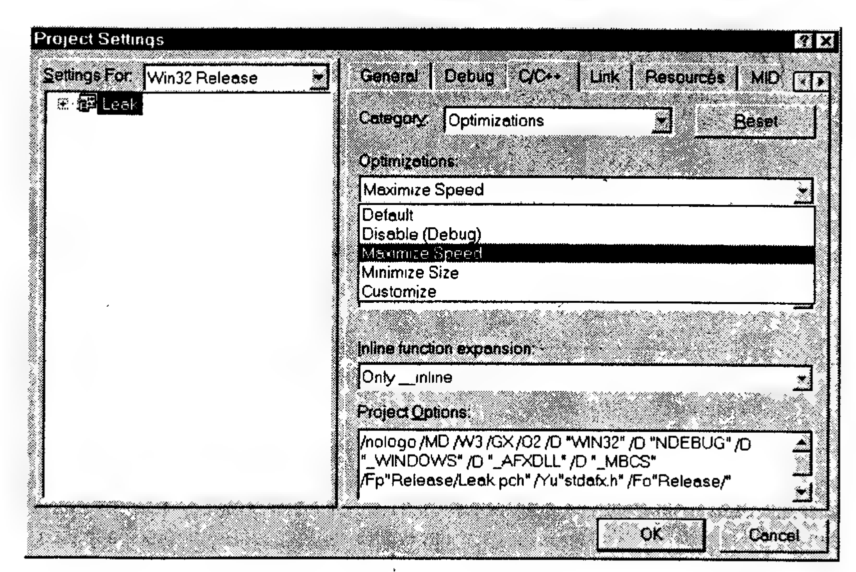


Рис. 24.5. Выбор требуемого типа оптимизации

Профилирование

Профилирование (profiling) приложения дает возможность обнаружить узкие места в тексте программы — те фрагменты исходного текста, которые замедляют работу приложения и заслуживают особого внимания разработчика. Нет смысла тратить время на оптимизацию некоторой процедуры, не зная, вызывается ли эта процедура настолько часто, чтобы оказать заметное влияние на скорость выполнения всей программы.

Другим назначением профилирования является контроль качества тестов. При помощи профилирования можно узнать, осуществляет ли выполняемый тест вызов и обработку возвращаемых результатов всех без исключения подпрограмм, существующих в данной программе, и выполняется ли при этом хотя бы однажды каждая строка исходного текста программы. Вы можете предполагать, что создали исчерпывающую тестовую последовательность входных данных, удовлетворяющую указанным выше требованиям, однако только профилирование приложения сможет это подтвердить.

Visual C++ содержит профилировщик, интегрированный со средой разработки, — нужно лишь уметь им пользоваться. Прежде всего, переопределите установленные в проекте опции с целью включения информации профилировщика. Для этого раскройте окно свойств проекта Project Settings, как это описано в предыдущем разделе, и щелкните на корешке вкладки Link. Установите флажок опции Enable Profiling (Разрешить профилирование) и щелкните на кнопке OK. Оттранслируйте проект. Компоновка приложения теперь будет проходить медленнее, поскольку при планировании профилировки нельзя выполнять инкрементное связывание, но вы всегда сможете вернуться к прежним установкам, собрав достаточно информации об особенностях функционирования своей программы. Выберите команду Build⇨Profile, и на экране раскроется диалоговое окно Profile, показанное на рис. 24.6.

Если вы не знаете точно, для чего предназначен каждый переключатель в этом диалоговом окне, щелкните на пиктограмме ? (знак вопроса) в правом верхнем углу окна, а затем щелкните на интересующем вас переключателе. Будет выведено краткое объяснение. (Если вам захочется добавить такой вид контекстной подсказки в свое приложение, обратитесь к главе 11.)

Рис. 24.6. Профилировщик может собирать различную информацию о вашем приложении

Профилировщик не пригодится вам в поисках причин возникновения ошибок в приложении, но поможет оценить качество используемых тестов и подскажет, над какими именно фрагментами приложения следует еще поработать. Эти возможности превращают его в ценнейший инструмент из числа имеющихся в арсенале профессиональных разработчиков программ. Выработайте в себе привычку хотя бы один раз выполнять профилировку для каждого из разрабатываемых приложений.

ГЛАВА 25

Как достичь повторного использования программных компонентов

В этой главе...

Преимущества создания повторно используемого программного текста

Использование Component Gallery

Специализированные мастеры

В наше время возможность повторного использоваиия программных компонентов стала больше чем просто данью моде. Она стала средством выживания для программистов, которые сталкиваются со сложными задачами написания сотен тысяч строк исходного текста работоспособной программы в предельно сжатые сроки. Visual C++ предоставляет множество вариантов повторного использования результатов работы, уже выполненной для вас другими программистами, включая мастеры AppWizard и ClassWizard и, конечно же, библиотеку классов MFC. Инструменты, о которых пойдет речь в этой главе, позволят вам стать одним из тех, кто создает программный текст на будущее. Этот текст готов к повторному использованию. Его можно будет легко и быстро помещать в программы, создаваемые вашими коллегами и вами.

Преимущества создания повторно используемого программного текста

Если перед вами поставлена определенная задача, полезно будет проанализировать, возможно ли повторно использовать уже созданный кем-то исходный текст, диалоговые окна или другие компоненты программы с тем, чтобы упростить и ускорить выполнение работы. И уж если вы уверены в поставщике программных компонентов, которые собираетесь использовать в собственной разработке, то чем больше этого материала будет включено в программу, тем лучше. Все это привело к формированию особого сектора рынка — рынка повторно используемых компонентов программ.

Фактически существует два рынка: формальный, формируемый поставщиками фрагментов проектов и компонентов, таких как элементы управления и шаблоны, и неформальный, существующий внутри многих больших компаний, отделы которых создают повторно используемые элементы программ для дальнейшего их применения в собственных будущих разработках. Некоторые компании даже имеют фонд стимулирования повторного использования, из которого оплачивается время, затраченное программистами на обеспечение повторного использования создаваемых ими элементов, или выплачиваются вознаграждения тем работникам, чьи элементы были повторно использованы кем-либо в компании. Если такого в вашей компании нет, возможно, вам стоило бы попробовать: повторное использование может сберечь до 60% средств, расходуемых на создание программного обеспечения, но эта потенциальная экономия превратится в реальную только в том случае, если разработчики действительно будут заинтересованы возможностью повторного использования создаваемых ими компонентов и в компании приняты правила поощрения применения подобных компонентов.

Большинство из тех, кто впервые обращаются к теме повторного использования, полагают, что повторно можно использовать только исходный текст, однако существуют и другие элементы проектов, повторное использование которых сэкономит гораздо больше времени, чем простое дублирование фрагментов программного кода. Ниже перечислены такие элементы.

- **Модель.** Парадигма *документ/представление*, обсуждаемая в главе 4, представляет собой классический пример модели, которая повторно используется при разработке многих проектов.
- **Ресурсы интерфейса.** Можно повторно использовать элементы управления, пиктограммы, меню, панели инструментов или же целиком диалоговые окна, сокращая затраты времени на переобучение пользователей, равно как и затраты времени на повторную разработку.
- **Настройка среды разработкн.** Касается это настройки компоновщика или же компоновки панелей инструментов, но рабочая среда должна быть для вас удобна и настроена в соответствии с вашими предпочтениями. Это приведет к сокращению времени разработки каждого следующего проекта благодаря повторному использованию решений, найденных прежде.

- **Документация**. Как вы могли прочесть в главе 11, текст справочных данных по стандартным командам, подобным File⇨Open, генерируется мастером AppWizard автоматически. Однако можно повторно использовать и справочные тексты, подготовленные самостоятельно, сэкономив при этом еще больше времени.

Использование Component Gallery

Component Gallery (Галерея компонентов) — это одно из средств поддержки технологии повторного использования компонентов в среде Visual Studio. Библиотека Component Gallery предоставляет непосредственный доступ ко всему, что можно повторно использовать, начиная от классов и элементов управления OLE и заканчивая мастерами. Можно добавить в Component Gallery и разработанные самостоятельно компоненты. При инсталляции Visual Studio с опциями, заданными по умолчанию, в Component Gallery автоматически добавляется категория для новых приложений, которые будут созданы в процессе работы при помощи мастера AppWizard.

Добавление компонента в Gallery

Предположим, имеется диалоговое окно, которое часто используется в проектах. Можно всего один раз создать это диалоговое окно, добавить его в Component Gallery, а затем помещать его в новые проекты везде, где оно потребуется. Чтобы получить наглядное представление о том, как это происходит на практике, выполните следующие операции.

1. Создайте с помощью мастера AppWizard новый проект, присвоив ему имя App1. Для этого достаточно на первом этапе настройки сразу щелкнуть на кнопке Finish, принимая все предложенные AppWizard установки по умолчанию, а затем щелкнуть на кнопке OK для завершения создания проекта.
2. Добавьте в проект новое диалоговое окно. Для этого выберите команду Insert⇨Resource и сделайте двойной щелчок на элементе Dialog в раскрывшемся диалоговом окне.
3. Используя технологию, описанную в главе 2, создайте диалоговое окно, показанное на рис. 25.1, присвоив ему идентификатор ресурса IDD_NAMEDLG.
4. Пока диалоговое окно остается в фокусе, раскройте окно мастера ClassWizard и выберите создание нового класса. Присвойте ему имя CNameDlg.
5. Закройте окно ClassWizard.
6. На панели ClassView щелкните правой кнопкой мыши на элементе CNameDlg и выберите из раскрывшегося контекстного меню команду Add To Gallery.

Хотя может сложиться впечатление, что ничего не произошло, тем не менее класс CNameDlg и связанный с ним ресурс были добавлены в Component Gallery. Сверните окно Visual Studio и проанализируйте содержимое жесткого диска, начиная с окна My Computer (Мой компьютер) и вплоть до папки C:\Program Files\Microsoft VisualStudio\Common\MSDev98\Gallery (если вы установили Visual C++ в другую папку, перейдите к соответствующей папке MSDev98 и продолжайте с этого места). Убедитесь, что в папке Gallery появилась новая папка App1 (рис. 25.2).

Сделайте двойной щелчок на папке App1, и вы увидите, что она содержит один файл с именем Name Dlg.ogx (рис. 25.3). Расширение .ogx означает, что данный файл содержит компонент из Component Gallery.

Рис. 25.1. Создание диалогового окна с целью его добавления в `Component Gallery`

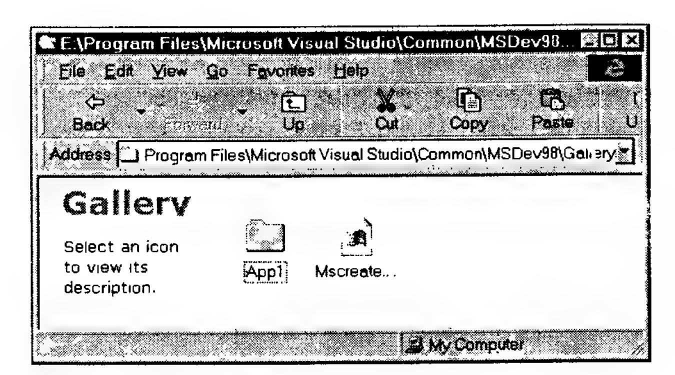


Рис. 25.2. При добавлении в `Component Gallery` *класса в качестве имени папки используется имя соответствующего проекта*

Использование в проектах компонентов из `Component Gallery`

Теперь, после добавления в `Component Gallery` ресурса и соответствующего ему класса, логично будет на следующем шаге создать еще один проект, который и будет этот ресурс использовать. Создайте с помощью мастера `AppWizard` еще одно приложение с именем `App2`, для чего снова щелкните на кнопке Finish, приняв все предложенные мастером установки по умолчанию, а затем завершите создание нового проекта, щелкнув на кнопке OK.

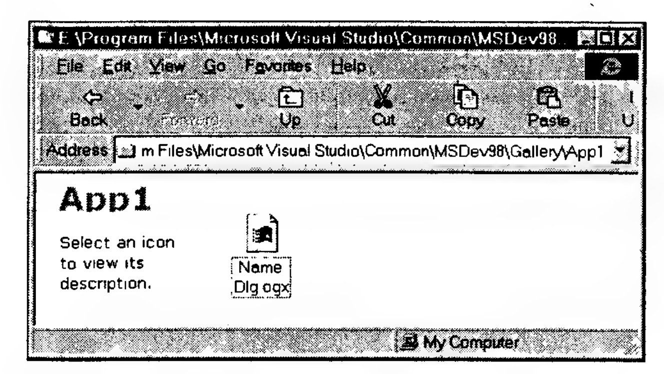


Рис. 25.3. Имя файла, содержащего элемент для `Component Gallery`*, создается на основании имени класса*

Щелкните на корешке вкладки ClassView и раскройте дерево классов проекта App2. Их должно быть шесть: CAboutDlg, CApp2App, CApp2Doc, CApp2View, CChildFrame и CMainFrame.

Выберите команду Project⇨Add To Project⇨Components and Controls (Проект⇨Добавить в проект⇨Компоненты и элементы управления). Раскроется диалоговое окно Components and Controls Gallery, показанное на рис. 25.4.

Сделайте двойной щелчок на папке App1, и вы вновь увидите файл Name Dlg.ogx. Сделайте на нем двойной щелчок. В раскрывшемся окне подтвердите ваше намерение вставить этот компонент в проект. Закройте диалоговое окно Gallery, щелкнув на кнопке Close.

Ещё раз взгляните на панель ClassView. В дерево классов добавлен новый класс CNameDlg. Щелкните на корешке вкладки FileView и убедитесь, что в проект добавлены файлы NameDlg.cpp и NameDlg.h. Переключитесь на вкладку ResourceView и убедитесь, что в проект добавлено диалоговое окно IDD_NAMEDLG. В приложении App2 этот ресурс можно использовать таким же образом, как и в приложении App1.

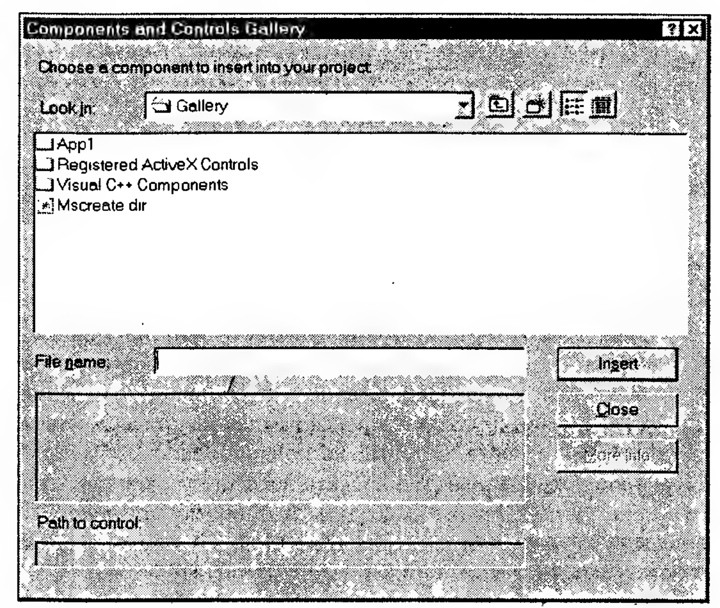


Рис. 25.4. В `Component Gallery` *компоненты распределены по папкам*

Возможности Component Gallery

Библиотеку Component Gallery можно использовать для размещения множества других типов компонентов, включая те, которые вы могли получить от друга или купить у третьего лица. Библиотека Component Gallery обеспечивает добавление, удалеиие, импорт и различные способы редактирования компонентов, зависящие от типа компонента, с которым вы работаете. Потратьте некоторое время на эксперименты с Component Gallery, и вы быстро оцените простоту ее использования.

На рис. 25.5 представлены компоненты, хранящиеся в папке Registered ActiveX Controls. Здесь присутствуют как ATL-, так и MFC-версии элемента управления Dieroll: элемент Dieroll Class (ATL) был создан в главе 21, а элемент Dieroll Control (MFC) — в главе 17. Перед тем как была снятая копия экрана, использованная для создания рис. 25.5, был выделен компонент Grid Control и сделан щелчок на кнопке More Info. К компонентам может прилагаться справочный файл, доступ к которому осуществляется после щелчка на кнопке More Info.

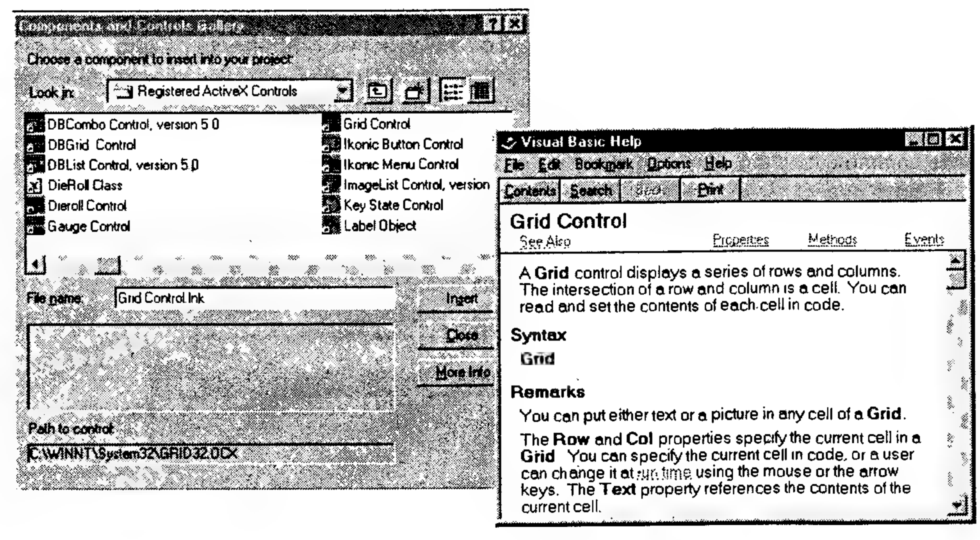


Рис. 25.5. В окне Component Gallery доступны все элементы управления ActiveX

Специализированные мастеры

Мастер AppWizard является замечательным инструментом, позволяющим легко и быстро создавать основу для новых проектов. Однако именно по причине своей универсальности AppWizard делает достаточно много предположений о функциоиальных возможностях, которые вы намерены заложить в новый проект. Иногда вам может потребоваться создать специфический вариант заготовки проекта, который не поддерживается обычным AppWizard, настроенным по умолчанию. Если данный тип проекта потребуется вам лишь однажды, вероятно, следует просто приступить к работе и создать этот проект вручную. Однако, если вам постоянно приходится иметь дело с проектами такого типа, имеет смысл создать специализированный вариант мастера AppWizard.

Специализированный вариант мастера AppWizard можно создать тремя способами, используя в качестве исходного материала этапы настройки существующего мастера AppWizard, используя в качестве исходного материала существующий проект или же начать все с нуля.

Тем не менее независимо от выбранного подхода создание специализированного мастера AppWizard может быть достаточно сложной задачей, требующей от вас понимания языка и умения писать файлы сценариев, используя для этого макросы и команды, предоставленные для этих целей Visual C++.

Ниже описывается самый простой случай — создание мастера AppWizard, предназначенного для воспроизведения существующего проекта под другим именем. Выполните следующие действия.

1. Создайте проект обычным образом. Присвойте ему имя Original и на первом же шаге щелкните на кнопке Finish, принимая все предложенные AppWizard установки по умолчанию.
2. Отредактируйте диалоговое окно About Original так, как показано на рис. 25.6.
3. Выберите команду File⇨New и в раскрывшемся окне щелкните на корешке вкладки Projects. Выберите элемент Custom AppWizard, а в поле Project Name введите значение OrigWiz, как показано на рис. 25.7. Щелкните на кнопке OK.
4. На экране раскроется первое из двух диалоговых окон мастера Custom AppWizard, показанное на рис. 25.8. Установите переключатель An existing project (Существующий проект), указав на необходимость при построении нового мастера взять за основу проект, созданный нами раньше (см. п. 1 и 2). Не изменяйте имя нового мастера, предложенное по умолчанию. Щелкните на кнопке Next.

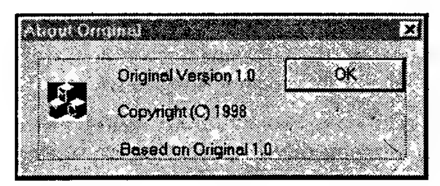


Рис. 25.6. Отредактированное окно About Original

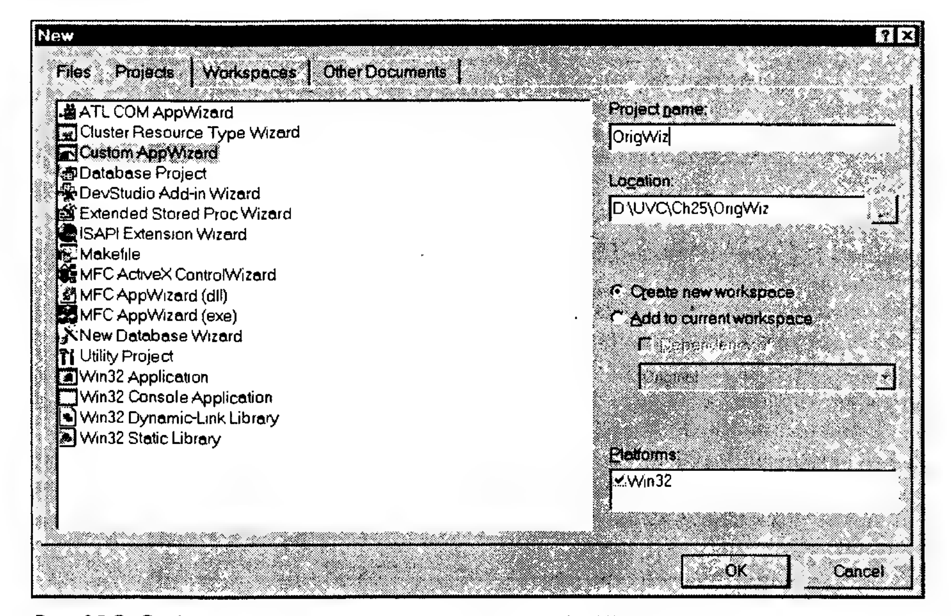


Рис. 25.7. Создание специализированного мастера AppWizard

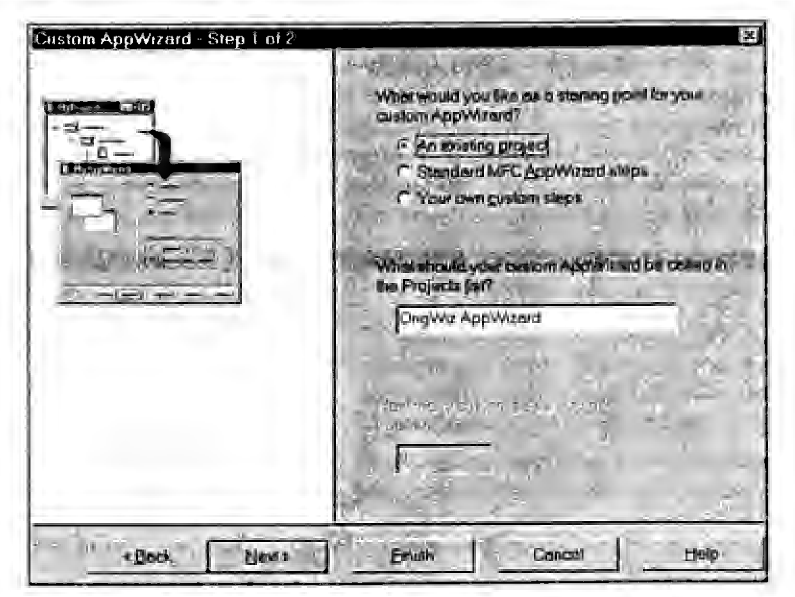


Рис. 25.8. Основой нового мастера служит существующий проект

5. Раскроется второе диалоговое окно мастера Custom AppWizard. Укажите файл существующего проекта, который будет взят за основу нового мастера. Щелкните на кнопке Finish.
6. Раскроется диалоговое окно New Project Information (Информация о новом проекте), показанное на рис. 25.9, в котором будут приведены все сделанные установки. Щелкните на кнопке OK.

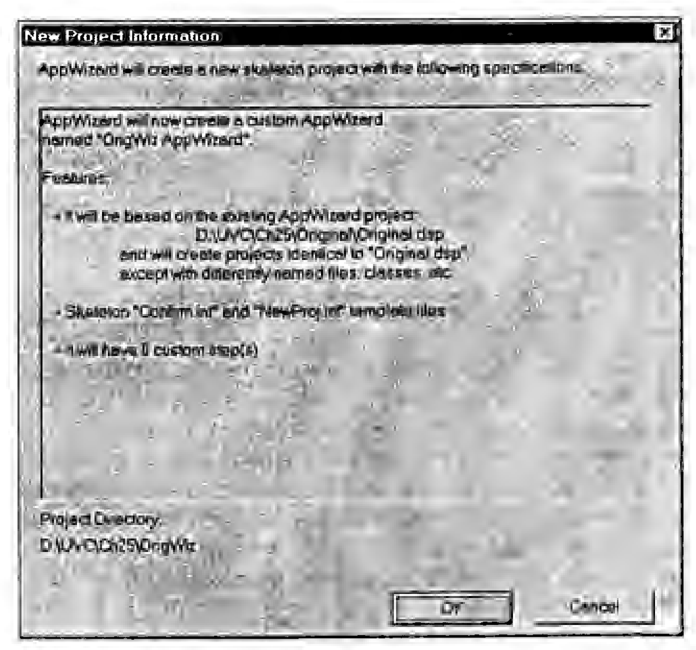


Рис. 25.9. Новый специализированный мастер AppWizard будет создавать копию проекта Original, но под другим именем

Теперь можно было бы продолжить работу над проектом OrigWiz, которая в большинстве случаев состояла бы в добавлении программного текста. Но поскольку это всего лишь пример, сразу же оттранслируйте проект.

Для запуска вновь созданного мастера AppWizard еще раз выберите команду File⇨New и щелкните на корешке вкладки Projects. Как видно из рис. 25.10, новый мастер OrigWiz внесен в расположенный в левой части окна список доступных типов проектов. Выберите его и введите значение App3 в качестве имени нового проекта. Щелкните на кнопке OK.

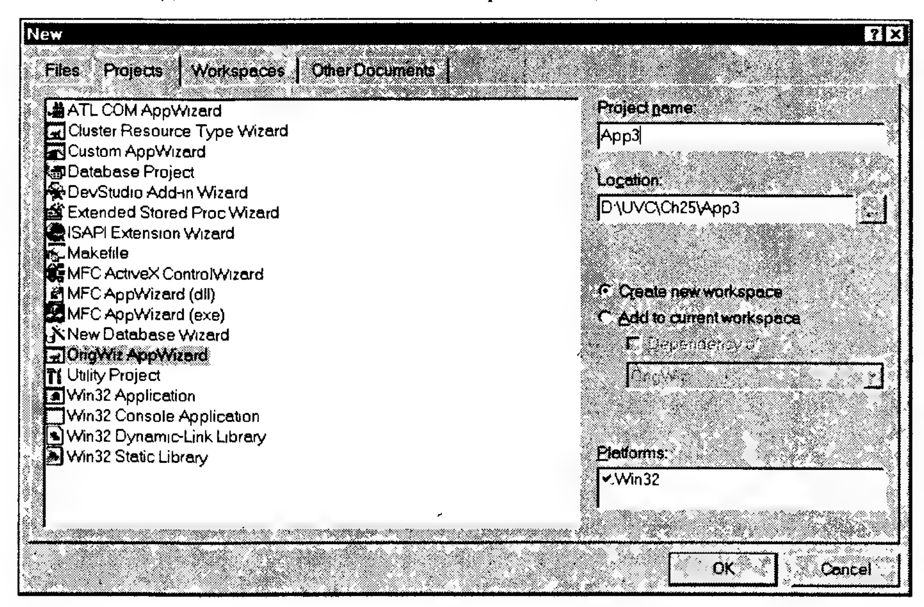


Рис. 25.10. Новый специализированный мастер AppWizard добавлен к существующему списку мастеров

На заметку

При компиляции нового специализированного мастера AppWizard в среде Visual Studio все созданные файлы будут помещены в папку C:\Program Files\Microsoft VisualStudio\Common\MSDev98\Template. При создании очередного нового проекта созданный вами специализированный мастер AppWizard будет помещен в список существующих типов проектов. Для удаления специализированного мастера AppWizard удалите его файлы с расширениями .awx и .pdb из каталога C:\Program Files\Microsoft Visual Studio\Common\MSDev98\Template.

На рис. 25.11 показано одно из заданий, которое в реальном проекте следует выполнить перед компиляцией нового мастера AppWizard. В данном случае задание предусматривает подготовку текста для диалогового окна New Project Information. Щелкните на кнопке OK.

Посмотрите на имена классов и тексты программ — проект App3 ничем не отличается от ранее созданных в этой главе проектов, для которых принимались все установки, предлагаемые мастером AppWizard по умолчанию. Однако в данном случае не потребовалось последовательно переключать окна AppWizard. Переключитесь на панель ResourceView и откройте для редактирования ресурс IDD_ABOUTBOX. Как видно из рис. 25.12, диалоговое окно About содержит добавленный вами текст (Based on Original 1.0), но имя приложения в верхней строке правильно изменено на App3. Этот мастер кое-что все-таки умеет делать.

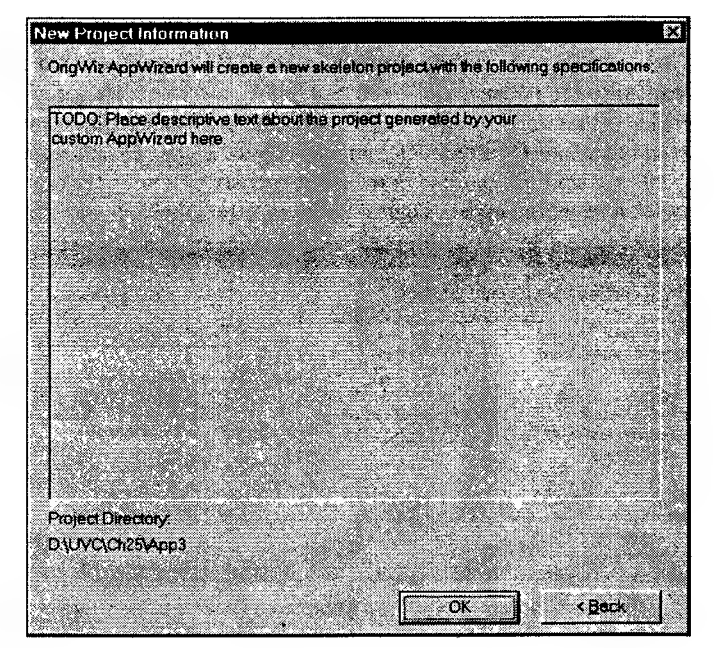


Рис. 25.11. Необходимо ввести текст, который будет помещен в диалоговое окно New Project Information

Когда вы создаете мастер с использованием существующего проекта, все имеющиеся в нем классы, ресурсы и исходный текст копируются в новые проекты, создаваемые этим мастером. Экономия времени очевидна.

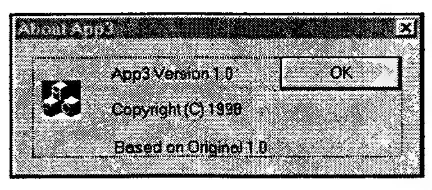


Рис. 25.12. Измененное окно About *специализированный мастер* AppWizard *скопировал в новый проект*

Кроме того, можно создавать такие специализированные мастера AppWizard, которые будут использовать диалоговые окна, предоставляя вам возможность устанавливать опции и т.д. Однако прежде нужно научиться создавать другие типы мастеров, о которых речь шла в главе 12. Также вам следовало бы приобрести опыт генерации с помощью мастеров множества разнообразных приложений. Это даст возможность понять специфику выполняемой мастерами AppWizard работы. И когда будете готовы, откройте в электронной справке раздел *Creating Custom AppWizards* и принимайтесь за работу.

На протяжении всей этой книги подчеркивается важность повторного использования созданных другими программистами моделей, классов, исходного текста, элементов управления, диалоговых окон и других частей проекта. В этой главе было рассказано о двух простых способах придания исходному тексту программы формы, пригодной для повторного использования. Это может принести реальную пользу вашим клиентам или коллегам, сохраняя драгоценное время, отпущенное на разработку. К тому же это поможет внести в работу элемент творчества, освобождая от выполнения рутинных задач, скажем, таких, как создание диалогового окна и связывание его с классом.

ГЛАВА 26

Исключения, шаблоны и последние модификации C++

В этой главе...

Работа с исключениями

Шаблоны

Библиотека стандартных шаблонов

Использование пространств имен

Обзор новых ключевых слов и типов данных

Язык C++ постоянно развивается и совершенствуется. Новыми мощными средствами, которые совсем недавно были добавлены в C++, являются исключения и шаблоны. В стандарт языка также включено несколько новых ключевых слов. Хотя большинство программистов предпочтет отложить изучение этих инноваций на потом, полагая, что сначала нужно приобрести хотя бы полугодичный опыт работы в Visual C++, мы настоятельно рекомендуем не откладывать их освоение в долгий ящик. Указанные новые концепции едва ли сложнее материала, который уже рассмотрен нами в этой книге, но они помогут реально расширить функциональные возможности создаваемых вами программ.

Работа с исключениями

Создавая программы на языке Visual C++, вы рано или поздно встретитесь с проблемой обработки ошибок, которая, как мне кажется, не имеет решения, пригодного на все случаи жизни. Допустим, вы пишете функцию, возвращающую числовое значение, и при этом пытаетесь найти способ передачи сообщения об ошибке. В некоторых случаях для этого подойдет одно значение возвращаемого параметра, к примеру 0 или -1, указывающее на возникшие проблемы. В других ситуациях может показаться, что передать уведомление об ошибке просто невозможно. Предположим, что в качестве индикатора возникновения ошибки используется некоторое конкретное значение, причем начало программы выглядит следующим образом:

```
while (somefunction(x))
{
        for (int i=0; i<limit; i++)
        {
                y = someotherfunction(i);
        }
}
```

Однако затем, поразмыслив, вы, возможно, придете к заключению, что если `someotherfunction()` возвратит значение –1, то переход к следующему шагу цикла `while` теряет смысл и следует просто выйти из него. В результате отредактированный текст программы будет выглядеть следующим образом:

```
int timetostop = 0;
while (somefunction(x)&& !timetostop)
{
        for (int i=0; i<limit && !timetostop; i++)
        {
                if ( (y = someotherfunction(i)) == -1)
                timetostop = 1;
        }
}
```

Получилось неплохо, однако разобраться сходу в таком тексте трудновато. При необходимости отслеживать возникновение ошибок в двух, трех или более функциях текст программы может стать чрезвычайно запутанным.

Для устранения проблем именно такого типа и предназначены *исключения*. Механизм исключений позволяет программам сообщать друг другу о возникновении серьезных и неожиданных проблем. Большинство исключений реализуется в программе тремя следующими фрагментами кода.

- Блок `try` отмечает тот участок текста программы, в котором потенциально возможно возникновение ошибки.

- Блок catch следует непосредственно за блоком try и содержит операторы обработки обнаруженной ошибки.
- Оператор throw используется для передачи сообщения об ошибке в вызывающую часть программы — принято говорить, что оператор throw *выбрасывает исключение*.

Обработка простых исключений

Механизм, используемый для обработки исключительной ситуации, в действительности чрезвычайно прост. Фрагмент текста программы, в котром могут возникнуть ошибки, следует поместить внутрь блока try. Затем создается блок catch, который будет выполнять функции обработки ошибок. Если в блоке try (или в том фрагменте программы, который *вызывается* в блоке try), будет сгенерировано исключение (управление будет передано оператору throw), то дальнейшее выполнение блока try немедленно прекратится и программа передаст управление блоку catch.

Известно, что одним из тех мест в программах, где можно ожидать возникновения неприятностей, является распределение памяти. В листинге 26.1 приведен текст небольшой демонстрационной программы, которая распределяет некоторое количество памяти и затем сразу же ее освобождает. Поскольку выделение памяти может закончиться неудачно, операторы выделения памяти помещены в блок try. Если указатель, возвращаемый после распределения памяти, будет содержать значение NULL, то в блоке try будет вызвано исключение. В данном случае параметром оператора throw (параметром исключения) является строковая константа.

На заметку Приводимые в этой главе примеры программ являются приложениями, осуществляющими вывод на консоль, т.е. они могут быть запущены из командной строки DOS и не имвют графического интерфейса. Благодаря этому они имеют достаточно малые размеры и текст их полностью помещается е приводимых листингах. Посмотреть на их работу можно, создав консольное приложение, как описано в главе 28, добавив файл в проект и поместив е этот файл тексты программ, приводимые в листингах.

Листинг 26.1. Файл EXCEPTION1.CPP — обработка простых исключений

```
#include <iostream.h>

int main()
{
        int* buffer;

        try
        {
                buffer = new int[256];

                if (buffer == NULL)
                        throw "Memory allocation failed!";
                else
                        delete buffer;
        }
        catch(char* exception)
        {
                cout << exception << endl;
        }

        return 0;
}
```

Когда в программе выбрасывается исключение, выполнение программы продолжается с первой строки блока catch. (Оставшаяся в блоке try часть текста программы не выполняется.) В программе из листинга 26.1 единственный оператор блока catch просто выводит сообщение на консоль, после чего выполняется оператор return и программа завершает работу.

Если выделение памяти пройдет успешно, будет выполнен весь блок try и отведенная под buffer память будет возвращена операционной системе. В этом случае весь блок catch будет пропущен и программа перейдет непосредственно к выполнению оператора return.

На заметку

На самом деле в блоке catch не только продолжается выполнение программы. Этот блок еще и перехватывает параметр исключения, вызванного в программе. К примеру, в программе из листинга 26.1 перехватываемый параметр исключения помещен в круглые скобки непосредственно после ключеаого слова catch. Все это очень похоже на передачу аргумента функции-члену. В данном случае тип этого параметра — char*, а имя — exception.

Объекты исключений

Вся прелесть механизма исключений языка C++ заключается в том, что посылаемый при вызове исключения параметр может быть объектом любой структуры. Например, вполне реально создать специальный класс для определенных типов исключений, которые могут генерироваться в ваших программах. В листинге 26.2 приведен текст программы, в которой определяется универсальный класс исключений с именем MyException. В случае неудачного завершения процедуры выделения памяти главная программа создаст объект этого класса и вызовет исключение, параметром которого и будет этот объект. Блок catch перехватит параметр — объект класса MyException, вызовет метод GetError() этого объекта и выберет с его помощью строку сообщения об имевшей место ошибке, а затем отобразит эту строку на экране.

Листинг 26.2. Файл EXCEPTION2.CPP — создание класса исключений

```
#include <iostream.h>

class MyException
{
protected:
        char* m_msg;

public:
        MyException(char *msg) { m_msg = msg; }
        ~MyException(){}
        char* GetError() {return m_msg; };
};

int main()
{
        int* buffer;

        try
        {
                buffer = new int[256];
                if (buffer == NULL)
                {
                        MyException* exception =
                                new MyException("Memory allocation failed!");
                        throw exception;
                }
                else
                        delete buffer;
```

```
        }
        catch(MyException* exception)
        {
                char* msg = exception->GetError();
                cout << msg << endl;
        }

        return 0;
}
```

Объект исключения может быть предельно прост, определяя лишь целочисленный код ошибки, либо достаточно сложен и представлять собой тщательно разработанный класс. Библиотека MFC предоставляет набор классов исключений, который включает базовый класс CException и несколько производных от него классов. Абстрактный класс CException имеет конструктор и три метода: Delete(), удаляющий объект исключения, GetErrorMessage(), возвращающий строку, описывающую имевшее место исключение, и ReportError(), который выводит сообщение.

Размещение блока catch

Программный блок catch не обязательно должен располагаться в той же функции, в которой генерируется исключение. Когда исключение выброшено, система начинает просматривать стек вызова функций в поиске адреса ближайшего блока catch. Если блок catch не найден в функции, вызвавшей исключение, система продолжает поиск блока в функции, вызвавшей эту функцию. Поиск будет продолжаться до тех пор, пока не будет исчерпан стек вызовов функций. Если при этом не будет найден блок обработки исключений, выполнение программы прекратится.

В листинге 26.3 приведен текст короткой программы, демонстрирующей этот механизм. В этой программе исключение возбуждается в функции AllocateBuffer(), а перехватывается в функции main(), вызвавшей функцию AllocateBuffer().

Листинг 26.3. Файл EXCEPTION3.CPP — перехват исключения вне вызвавшей его функции

```
#include <iostream.h>

class MyException
{
protected:
        char* m_msg;

public:
        MyException(char *msg) { m_msg = msg;}
        ~MyException(){}
        char* GetError() {return m_msg;}
};

class BigObject
{
private:
        int* intarray;
public:
        BigObject() {intarray = new int[1000];}
        ~BigObject() {delete intarray;}
};

int* AllocateBuffer();
```

```
int main()
{
        int* buffer;

        try
        {
                buffer = AllocateBuffer();
                delete buffer;
        }
        catch (MyException* exception)
        {
                char* msg = exception->GetError();
                cout << msg << endl;
        }

        return 0;
}

int* AllocateBuffer()
{
        BigObject huge;
        float* floatarray = new float[1000];
        int* buffer = new int[256];

        if (buffer == NULL)
        {
                MyException* exception =
                        new MyException("Memory allocation failed!");
                throw exception;
        }

        delete floatarray;
        return buffer;
}
```

Когда в функции AllocateBuffer() выбрасывается исключение, остальные операторы в теле функции не выполняются. По этой причине динамически созданный массив floatarray не будет удален[1]. Объект BigObject, размещенный в стеке, выйдет из области видимости и будет автоматически вызван его деструктор, удаляющий переменную-член intarray, созданную в конструкторе объекта с помощью оператора new. Эту важную концепцию следует глубоко усвоить: объекты, размещенные в стеке, будут удалены с помощью их деструкторов в процессе освобождения стека. С объектами, память под которые была выделена из кучи, этого не произойдет — об их удалении должна позаботиться созданная вамн программа. К примеру, функция AllocateBuffer() должна содержать операторы, удаляющие массив floatarray, до оператора вызова исключения, как показано ниже:

```
if (buffer == NULL)
        {
                MyException* exception =
                        new MyException("Memory allocation failed!");
                delete floatarray;
                throw exception;
        }
```

[1] Удаление динамического программного объекта есть не что иное, как высвобождение занятой им памяти и возвращение ее в кучу, которая находится под управлением операционной системы. — *Прим. ред.*

Во многих случаях при работе с исключениями использование объектов с тщательно продуманными деструкторами помогает избежать дублирования операторов в основном тексте программы. Если вы работаете с объектами, память для которых выделяется из общей кучи, может понадобиться перехватить одно исключение и вновь сгенерировать следующее просто для того, чтобы удалить отведенную под эти объекты память. Проанализируйте представленный в листинге 26.4 текст программы, в которой исключение выбрасывается в самом конце промежуточной функции, вложенной в ту функцию, в которой имеется блок catch.

Листинг 26.4. Файл EXCEPTION4.CPP — иллюстрация работы механизма анализа стека

```
#include <iostream.h>

class MyException
{
protected:
        char* m_msg;

public:
        MyException(char *msg) { m_msg = msg;}
        ~MyException(){}
        char* GetError() {return m_msg;}
};

class BigObject
{
private:
        int* intarray;
public:
        BigObject() {intarray = new int[1000];}
        ~BigObject() {delete intarray;}
};

int* AllocateBuffer();
int* Intermediate();

int main()
{
        int* buffer;

        try
        {
                buffer = Intermediate();
                delete buffer;
        }
        catch (MyException* exception)
        {
                char* msg = exception->GetError();
                cout << msg << endl;
        }

        return 0;
}

int* Intermediate()
{
        BigObject bigarray;
        float* floatarray = new float[1000];
        int* retval = AllocateBuffer();
        delete floatarray;
```

```
        return retval;
}

int* AllocateBuffer()
{
        int* buffer = new int[256];

        if (buffer == NULL)
        {
                MyException* exception =
                        new MyException("Memory allocation failed!");
                throw exception;
        }

        return buffer;
}
```

В данном случае при вызове исключения выполнение функции AllocateBuffer() немедленно прекращается и начинается анализ стека. Поскольку в функции Intermediate() блок catch отсутствует, ее выполнение будет прекращено сразу же после вызова функции AllocateBuffer(). Освобождение памяти, отведенной под floatarray, выполнено не будет, но деструктор для объекта bigarray будет вызван. В листинге 26.5 приведен метод решения подобной проблемы.

Листинг 26.5. Повторный вызов исключения

```
int* Intermediate()
{
        BigObject bigarray;
        float* floatarray = new float[1000];
        int* retval = NULL;
        try
        {
                retval = AllocateBuffer();
        }
        catch (MyException* e)
        {
                delete floatarray;
                throw;
        }
        delete floatarray;
        return retval;
}
```

Пересмотренная версия функции Intermediate() перехватывает исключение, что дает ей возможность удалить выделенную под floatarray память, а затем повторно выбрасывает исключение, передавая его на обработку в вызывающую функцию. (Обратите внимание, что в новом операторе throw отсутствует параметр — повторно выбросить можно только то исключение, которое было перед этим перехвачено.) Ниже приведено несколько замечаний, касающихся этой пересмотренной функции.

- Оператор высвобождения памяти, выделенной под floatarray, дублируется.
- Объявление переменной retval вынесено за пределы блока try, так что эта переменная остается в области видимости и после выполнения блока try.
- Переменная retval теперь должна быть проинициализирована значением по умолчанию.

В результате всех наших последних манипуляций текстом программы экземпляр класса BigObject с именем bigarray корректно и просто удаляется безо всяких забот с нашей сторо-

ны, поскольку соответствующий деструктор вызывается автоматически. Это выполняется независимо от того, какой функцией он был создан в памяти или в каком месте программы вызвано исключение. Поэтому при разработке программы, работающей с исключениями, рекомендуется помещать все объекты, память под которые выделяется из кучи, в классы типа BigObject. Класс BigObject использует *управляемый указатель* (managed pointer); когда объект класса BigObject, скажем, bigarray, выходит за пределы области видимости, память, на которую он указывает, автоматически освобождается. Исключительно гибкий подход к управляемым указателям описан в конце раздела *Шаблоны,* ниже в этой же главе.

Обработка исключений разных типов

Поскольку очень часто в определенных фрагментах программы возникает необходимость использовать более одного типа исключений[2], допускается с одним блоком try использовать несколько различных блоков catch. К примеру, может потребоваться перехватывать исключения как типа CException, так и типа char*. Поскольку блок catch обрабатывает только один конкретный тип передаваемого ему параметра исключения, потребуется два различных блока для обработки параметров исключений типа CException и типа char*. Имеется возможность так определить блок catch, что он будет перехватывать любой тип исключений, которые до данного момента еще не были перехвачены. Это происходит, если в качестве его аргумента вместо параметра конкретного типа помещается многоточие (...). Проблема при использовании такого многоцелевого блока catch заключается в том, что получить доступ к параметру перехваченного исключения будет невозможно и, следовательно, придется обрабатывать исключение некоторым общим способом.

В листинге 26.6 представлен текст программы, в которой, в зависимости от введенных пользователем данных, выбрасываются исключения трех различных типов. (В реальной программе не следует использовать исключения при обработке ошибок во введенных пользователем данных. Это слишком медленный механизм, и проверка введенной пользователем информации, как правило, должна быть реализована посредством других, более эффективных методов.)

Когда вы запустите программу, на экран будет выведено предложение ввести любое число от 4 до 8, за исключением 6. Если вы введете число, меньшее 4, программа выбросит исключение MyException; если вы введете число, большее 8, программа выбросит исключение char*; и, наконец, если вы введете число 6, программа выбросит исключение целочисленного типа, значение которого равно введенному числу.

Хотя все исключения выбрасываются из функции GetValue(), перехват их производится в функции main(). Блок try в функции main() связан с тремя блоками catch. Первый блок перехватывает объект MyException, второй блок перехватывает объект char*, а третий перехватывает любое другое исключение.

На заметку

Как и в случае с конструкцией if–else, порядок размещения блоков catch оказывает существенное влияние на работу программы. Блоки catch, обрабатывающие конкретные типы объектов, всегда должны размещаться первыми. Если бы в примере из листинга 26.6 блок catch(...) был первым, никакой другой из блоков никогда бы не выполнялся. Причина в том, что блок catch(...) является самым общим из всех возможных и перехватывает исключение любого типа. В данном случае (как и в большинстве прочих случаев) блок catch(...) используется только для перехвата исключений, которые не были перехвачены остальными блоками catch.

Листинг 26.6. Файл EXCEPTION6.CPP — использование нескольких блоков catch

```
#include <iostream.h>
```

[2] Тип исключения — это тип параметра в операторе throw. — *Прим. ред.*

```
class MyException
{
protected:
        char* m_msg;

public:
        MyException(char *msg) { m_msg = msg;}
        ~MyException(){}
        char* GetError() {return m_msg;}
};

int GetValue();

//----------------------------------------------------
int main()
{
        try
        {
                int value = GetValue();
                cout << "The value you entered is okay." << endl;
        }
        catch(MyException* exception)
        {
                char* msg = exception->GetError();
                cout << msg << endl;
        }
        catch(char* msg)
        {
                cout << msg << endl;
        }
        catch(...)
        {
                cout << "Caught unknown exception!" << endl;
        }

        return 0;
}

int GetValue()
{
        int value;

        cout << "Type a number from 4 to 8 (except 6):" << endl;
        cin >> value;

        if (value < 4)
        {
                MyException* exception =
                        new MyException("Value less than 4!");
                throw exception;
        }
        else if (value > 8)
        {
                throw "Value greater than 8!";
        }
        else if (value == 6)
        {
                throw value;
        }
        return value;
}
```

Устаревший механизм исключений

До того как операторы try, catch и throw были добавлены в Visual C++, в нем существовала примитивная форма обработки исключений, доступная как в языке С, так и в языке С++. Этот механизм был реализован с помощью макросов TRY, CATCH и THROW. Указанные макросы работают несколько медленнее, чем стандартный механизм исключений, и могут работать только с исключениями, которые являются объектами классов, производных от CException. Не используйте их в своих программах. Если у вас уже есть готовые программы, в которых они применялись, имеет смысл модифицировать их и перейти на новый механизм. В документации по Visual C++ есть полезный раздел, посвященный данной теме. Его можно найти, выполнив поиск по термину TRY .

Шаблоны

Весьма вероятно, что рано или поздно у вас возникнет идея разработать функцию (или класс), которая будет способна обрабатывать данные различных типов. Безусловно, создав несколько версий функции, всегда можно воспользоваться их перегрузкой или же использовать наследование для порождения нескольких различных классов от одного базового класса. Однако и в этих случаях дело все равно сводится к написанию *нескольких* различных функций или классов. Более привлекательна возможность создать "разумные" функции или классы, способные работать с любым типом данных, который будет передан ей на обработку. Способ решить эту, кажущуюся на первый взгляд неразрешимой, задачу существует. Для этой цели используются языковые конструкции, называемые *шаблонами*, о которых и пойдет речь в данном разделе.

Что такое шаблоны

Шаблон является эскизом или же схемой построения функции или класса. Вы создаете шаблон, как обычную функцию или класс, используя при описании объектов данных, которыми будет манипулировать окончательный вариант этой функции или класса, специальные "заместители", называемые *псевдопараметрами*. Описание шаблона всегда начинается с ключевого слова template, за которым следует список псевдопараметров, заключенный в угловые скобки, как показано ниже:

```
template<class Type>
```

Можно использовать сколько угодно псевдопараметров и называть их по своему усмотрению, но каждый из псевдопараметров должен начинаться с ключевого слова class и все они должны отделяться друг от друга запятыми, как показано ниже:

```
template<class Type1, class Type2, class Type3>
```

Существует два типа шаблонов: *шаблоны функций* и *шаблоны классов*. Ниже будет рассказано о том, как создавать и использовать оба типа шаблонов.

Создание шаблонов функций

Как мы уже знаем, шаблон функции начинается со строки template, за которой следует объявление функции, как показано в листинге 26.7. В строке template указываются типы аргументов, которые будут использоваться при вызове функции. Следовательно, в объявлении функции должно быть определено, как эти аргументы будут передаваться функции в качестве псевдопараметров. Каждый псевдопараметр, указанный в строке template, должен быть использован в

объявлении функции. Обратите внимание на тип Type1, стоящий непосредственно перед именем функции. Type1 — это держатель места, определяющий тип возвращаемого функцией значения, который будет изменяться в зависимости от способа использования шаблона.

Листинг 26.7. Основная форма шаблона функции

```
template<class Type1, class Type2>
Type1 MyFunction(Type1 data1, Type1 data2, Type2 data3)
{
        // Здесь разместите текст функции.
}
```

Разобраться в том, как шаблоны функций превращаются в реальные функции проще на примере. Таковым может быть функция Min(), способная работать с аргументами различных типов. В листинге 26.8 приведен текст короткой программы, в которой определяется шаблон для функции Min(), а затем к этой функции осуществляется обращение из функции main(). При выполнении программа отображает меньшее из значений данных любого типа, передаваемых функции Min() в качестве ее аргументов. Это возможно по той причине, что компилятор использует шаблон для создания на его основе реальных функций для каждого типа данных, сравниваемых в программе.

Листинг 26.8. Файл TEMPLATE1.CPP — использование типичного шаблона функции

```
#include <iostream.h>

template<class Type>
Type Min(Type arg1, Type arg2)
{
        Type min;

        if (arg1 < arg2)
                min = arg1;
        else
                min = arg2;

        return min;
}

int main()
{
        cout << Min(15, 25) << endl;
        cout << Min(254.78, 12.983) << endl;
        cout << Min(A, Z) << endl;

        return 0;
}
```

На заметку Обратите внимание на то, что в шаблоне функции Min() из листинга 26.8 тип данных Type используется не только в списках параметров и аргументов функции, но также для объявления локальных переменных в основном тексте функции. Это обеспечивает возможность использования псевдопараметров типа в шаблоне аналогично любому другому конкретному типу данных, такому как int или char.

Поскольку шаблонам функции присуща чрезвычайная гибкость, это часто может послужить причиной различных неприятностей. Так, например, при использовании шаблона Min() вы должны быть уверены, что для типов данных, передаваемых в качестве параметров, допускается операция сравнения. Если вы попытаетесь сравнить два класса, то программу не

удастся откомпилировать до тех пор, пока в классах не будут перегружены терминальные операторы < и >.

Еще один источник проблем заключается в некорректном вызове функции — когда аргументы, предоставляемые шаблону, используются не так, как вы предполагалн. Например, что произойдет, еслн в функцню main() из листинга 26.8 добавить приведенную ниже строку?

```
Cout << Min("APPLE", "ORANGE") << endl;
```

На первый взгляд, результатом выполнения приведенной выше строки будет вывод на экран значения APPLE. На самом же деле выполнение этой строки может привести, а может и не привести к ожидаемому результату. Почему? Да потому, что "APPLE" и "ORANGE" являются строковыми константами, представленными указателями на char. Программа будет успешно откомпилирована, при этом компилятор создаст версию функции Min(), которая будет сравнивать два указателя на тип char. Однако существует большая разница между сравнением двух указателей и сравнением данных, на которые они указывают. Если получится так, что строка "ORANGE" будет храниться по меньшему адресу, чем адрес строки "APPLE", то результатом работы данной версии функции Min() будет "ORANGE".

Избежать возникновения подобной проблемы можно путем использования при вызове Min() специальной *замещающей функции*, которая будет точно определять способ сравнения двух строковых констант. Когда вы включите в программу такую специальную функцию, компилятор использует ее вместо того, чтобы создавать функцию на основе шаблона. В листинге 26.9 приведен текст короткой программы, иллюстрирующей этот важный прием. Если программе необходимо сравнить две строки, то вместо вызова функции, созданной по шаблону, она будет использовать специальную замещающую функцию.

Листинг 26.9. Файл TEMPLATE2.CPP — использование специальной замещающей функции

```
#include <iostream.h>
#include <string.h>

template<class Type>
Type Min(Type arg1, Type arg2)
{
        Type min;

        if (arg1 < arg2)
                min = arg1;
        else
                min = arg2;

        return min;
}

char* Min(char* arg1, char* arg2)
{
        char* min;

  int result = strcmp(arg1, arg2);

        if (result < 0)
                min = arg1;
        else
                min = arg2;

        return min;
}
```

```
int main()
{
	cout << Min(15, 25) << endl;
	cout << Min(254.78, 12.983) << endl;
	cout << Min(A, Z) << endl;
	cout << Min("APPLE", "ORANGE") << endl;

	return 0;
}
```

Создание шаблонов классов

Точно так же, как создаются абстрактные функции на основе шаблонов функций, создаются абстрактные классы на основе шаблонов классов. Шаблон класса описывает класс, который, в свою очередь, описывает объект. Когда вы создаете шаблон класса, компилятор считывает этот шаблон и определяет класс. Затем вы создаете экземпляры объектов этого класса. Как видите, шаблоны классов определяют еще один уровень абстракции в концепции классов.

Шаблон класса определяется почти так же, как и шаблон функции, т.е. первой идет строка template, за которой следует объявление класса, как это показано в листинге 26.10. Обратите внимание, что, как и в шаблоне функции, в шаблоне класса абстрактные типы данных, которые использованы в качестве псевдопараметров в строке template, можно также использовать и в тексте описания класса, и при объявлении переменных-членов, указании типа возвращаемых значений, и для определения типа других объектов данных.

Листинг 26.10. Определение шаблона класса

```
Template<class Type>
class CMyClass
{
protected:
	Type data;

public:
	CMyClass(Type arg) { data = arg; }
	~CMyClass() {}
};
```

При создании по шаблону класса экземпляра объекта необходимо указать типы данных, которые будут подставлены вместо псевдопараметров шаблона. Например, для создания в программе объекта myClass класса CMyClass можно использовать следующий оператор:

```
CMyClass<int> myClass(15);
```

Эта строка создает объект класса CMyClass, который вместо абстрактного типа данных использует целочисленный. Если необходим класс для работы с вещественными числами, то объект такого класса можно создать следующим образом:

```
CMyClass<float> myClass(15.75);
```

Рассмотрим более сложный пример. Предположим, что необходимо создать класс, обеспечивающий хранение двух значений и содержащий метод их сравнения. В листинге 26.11 приведен текст программы, в которой реализован подобный класс. Прежде всего в программе определяется шаблон класса CCompare. В этом классе хранятся два значения, которые передаются конструктору при вызове. Класс также включает обычные конструктор и деструктор, а также методы для определения меньшего или большего из значений или проверки их на равенство.

Листинг 26.11. Файл TEMPLATE3.CPP — использование шаблона класса

```
#include <iostream.h>

template<class Type>
class CCompare
{
protected:
        Type arg1;
        Type arg2;

public:
        CCompare(Type arg1, Type arg2)
        {
                CCompare::arg1 = arg1;
                CCompare::arg2 = arg2;
        }

        ~CCompare() {}

        Type GetMin()
        {
                Type min;

                if (arg1 < arg2)
                        min = arg1;
                else
                        min = arg2;

                return min;
        }

        Type GetMax()
        {
                Type max;

                if (arg1 > arg2)
                        max = arg1;
                else
                        max = arg2;

                return max;
        }

        int Equal()
        {
                int equal;

                if (arg1 == arg2)
                        equal = 1;
                else
                        equal = 0;

                return equal;
        }
};

int main()
{
        CCompare<int> compare1(15, 25);
        CCompare<double> compare2(254.78, 12.983);
        CCompare<char> compare3(A, Z);
```

```
        cout << "THE COMPARE1 OBJECT" << endl;
        cout << "Lowest: " << compare1.GetMin() << endl;
        cout << "Highest: " << compare1.GetMax() << endl;
        cout << "Equal: " << compare1.Equal() << endl;
        cout << endl;

        cout << "THE COMPARE2 OBJECT" << endl;
        cout << "Lowest: " << compare2.GetMin() << endl;
        cout << "Highest: " << compare2.GetMax() << endl;
        cout << "Equal: " << compare2.Equal() << endl;
        cout << endl;

        cout << "THE COMPARE2 OBJECT" << endl;
        cout << "Lowest: " << compare3.GetMin() << endl;
        cout << "Highest: " << compare3.GetMax() << endl;
        cout << "Equal: " << compare3.Equal() << endl;
        cout << endl;

        return 0;
}
```

Главная программа создает на основе шаблона класса три различных объекта, один из которых оперирует целыми числами, другой использует вещественные числа, а третий хранит и сравнивает переменные символьного типа. Создав указанные три объекта класса `CCompare`, функция `main()` вызывает методы этих объектов с тем, чтобы вывести на экран информацию о данных, хранимых в каждом из них. На рис. 26.1 показан текст, выведенный данной программой на экран.

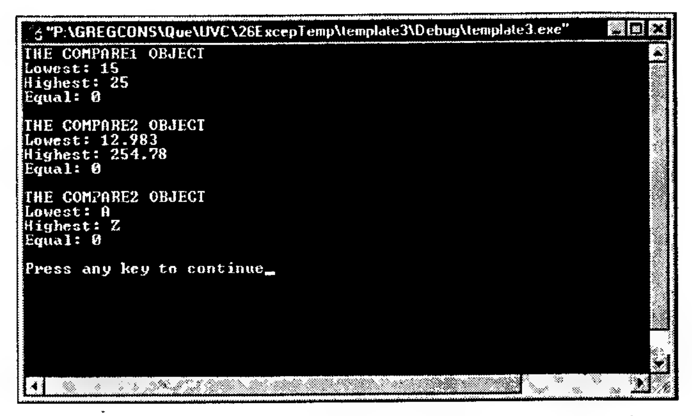


Рис. 26.1. Программа `Template3` создает на основе шаблона класса три различных объекта

В шаблоне класса, как и в шаблоне функции, можно указать сколько угодно псевдопараметров. В листинге 26.12 приведен текст шаблона класса, в котором используются два различных типа данных.

Листинг 26.12. Использование нескольких параметров в шаблоне класса

```
Template<class Type1, class Type2>
class CMyClass
{
protected:
        Type1 data1;
        Type2 data2;
```

```
public:
        CMyClass(Type1 arg1, Type2 arg2)
        {
                data1 = arg1;
                data2 = arg2;
        }

        ~CMyClass() {}
};
```

Для создания экземпляра объекта myClass класса CMyClass можно использовать, к примеру, следующую строку:

```
CMyClass<int, char> myClass(15,A);
```

И наконец, следует сказать, что в качестве псевдопараметров шаблона класса, наравне с типами-держателями места можно использовать и конкретные типы данных. Следует просто поместить в список псевдопараметров определенный тип данных наравне с прочими. В листинге 26.13 приведен текст небольшой программы, в которой экземпляр объекта создается на основе шаблона класса, использующего два абстрактных типа данных и один конкретный.

Листинг 26.13. Использование конкретных типов данных в качестве псевдопараметров в шаблоне класса

```
#include<iostream.h>

template<class Type1, class Type2, int num>
class CMyClass
{
protected:
        Type1 data1;
        Type2 data2;
        int data3;

public:
        CMyClass(Type1 arg1, Type2 arg2, int num)
        {
                data1 = arg1;
                data2 = arg2;
                data3 = num;
        }

        ~CMyClass() {}
};

int main()
{
        CMyClass<int, char, 0> myClass(15, A, 10);

        return 0;
}
```

Библиотека стандартных шаблонов

Не спешите приступать к созданию собственных шаблонов, реализующих связанные списки, бинарные деревья, сортировку и другие распространенные задачи. Вам, вероятно, будет приятно узнать, что это уже сделали другие. В Visual C++ включена библиотека стандартных шаблонов (STL — Standard Template Library), содержащая сотни шаблонов функций и клас-

сов, предназначенных для решения самых распространенных задач. Вам необходим стек целых чисел и стек вещественных чисел? Вам не придется создавать несколько классов, реализующих стек. Не нужно даже создавать шаблон класса, реализующего стек, — просто воспользуйтесь шаблоном стека из библиотеки STL. То же самое справедливо в отношении любой другой распространенной структуры данных или операции.

Выше в этой главе вы встречались с приложениями, в которых выполнялось распределение памяти из кучи (динамическое выделение памяти с помощью оператора new) и использовались исключения. В процессе выполнения в таких программах возникают проблемы, когда из-за вызова исключения пропускается выполнение оператора delete для освобождения выделенной из кучи памяти. Подобной проблемы можно избежать, если разместить в стеке объект, деструктор которого выполняет оператор delete для освобождения выделенной памяти. Библиотека STL предоставляет *управляемые указатели* подобного типа, называемые auto_ptr. Ниже приведено объявление класса auto_ptr:

```
template<class T>
        class auto_ptr
{
public:
        typedef T element_type;
        explicit auto_ptr(T *p = 0);
        auto_ptr(const auto_ptr<T>& rhs);
        auto_ptr<T>& operator =(auto_ptr<T>& rhs);
        ~auto_ptr();
        T& operator*() const;
        T *operator->() const;
        T *get() const;
        T *release() const;
};
```

Всякий раз, когда в программе создается указатель на int, float, Employee или на любой другой тип объекта, можно создать экземпляр класса auto_ptr, а затем использовать его как обычный указатель. Для примера рассмотрим следующий фрагмент программы:

```
// ...
        Employee* emp = new Employee(stuff);
        emp->ProcessEmployee;
        delete emp;
// ...
```

Когда выяснится, что функция ProcessEmployee() может выбросить исключение EmployeeException, то, скорее всего, этот фрагмент нужно будет заменить следующим:

```
// ...
        Employee* emp = new Employee(stuff);
        try
        {
                emp->ProcessEmployee;
        }
        catch (EmplyeeException e)
        {
                delete emp;
                throw;
        }
        delete emp;

// ...
```

Этот вариант довольно сложен для восприятия и имеет смысл заменить его, воспользовавшись объектом auto_ptr:

```
#include<memory>
// ...
	auto_ptr<Employee> emp (new Employee(stuff));
	emp->ProcessEmployee;
// ...
```

На первый взгляд, последний вариант фрагмента программы очень похож на первый, но работает он, как вторая версия. Независимо от того, завершится выполнение этого фрагмента нормально или оно будет прервано из-за генерацин исключения, объект emp выйдет из области видимости, и когда это произойдет, память, отведенная под объект класса Employee, будет автоматически освобождена. Не требуется никакнх блоков try или catch, к тому же вам вообще не нужно заботиться об освобождении памяти в данной подпрограмме.

Снова обратите внимание на функции, объявленные в шаблоне: конструктор, конструктор копирования, терминальный оператор взятия адреса (&), деструктор, терминальные операторы раскрытия ссылок (*) и (->), функции get() и release(). Все эти функции в своей совокупности призваны создать у вас впечатление полной идентичности работы с данным объектом и работы с обычным указателем.

Другие полезные стандартные шаблоны

Библиотека стандартных шаблонов особенно пригодится программистам, работающим с бнблиотекой Active Template Library, которые не используют MFC. Зачем стрелять из пушки по воробьям, приалекая всю мощь MFC, еслн все, что вам нужно, — это манипулировать строковыми переменными или просматривать таблицы и связанные списки? Вместо MFC вполне достаточно использовать версии этих распространенных структур из STL. Детальное их описание вы найдете в электронной справочной документации, а здесь мы перечислим только основные из большого списка полезных классов и функций.

- degue
- list
- map
- multimap
- set
- multiset
- vector
- basic_string
- stack
- swap
- min, max

Этим состав STL далеко не исчерпывается, но, познакомившись с ними поближе, вы поймете, сколько времени и средств можно сэкономить, используя шаблоны, особенно если их уже разработали ранее без вашего участия.

Использование пространств имен

Пространство имен определяет область видимости, в которой дублирование идентификаторов недопустимо. Например, после определения в программе глобальной переменной value вы определяете функцию, в которой под тем же именем используется локальная переменная. Поскольку эти две переменные находятся в разных пространствах имен, программа знает, что локальную переменную value следует использовать в теле функции, а глобальную переменную value — в любом другом месте.

Однако само по себе понятие пространства имен недостаточно широко, чтобы охватить некоторые очень деликатные вопросы. Примером тому являются одинаковые имена во внешних классах или библиотеках. Программисты сталкиваются с этой проблемой, используя несколько внешних файлов в одном проекте. Каждая из внешних переменных или функций должна иметь уникальное имя, отличное от имен других внешних функций и переменных. Во избежание возникновения подобных проблем поставщики программного обеспечения из других фирм часто добавляют приставки или суффиксы к именам переменных и функций, пытаясь таким способом снизить вероятность конфликта имен с компонентами других поставщиков.

Безусловно, корифеи C++ уже нашли способ решить проблемы, связанные с пространствами имен. (Иначе нам просто нечего было бы здесь обсуждать.) Таким радикальным решением является использование именованных пространств имен, которые определяются программистом. О них и пойдет речь в этом разделе.

Определение именованных пространств имен

Для использования определяемых пользователем пространств имен в язык C++ было введено ключевое слово namespace. В своей простейшей форме именованное пространство имен не отличается от структуры или класса. Определение именованного пространства имен начинается с ключевого слова namespace, за которым следует имя пространства имен и объявление идентификаторов, которые будут доступны в области видимости данного пространства имен.

В листинге 26.14 дано определение именованного пространства имен. Пространство имен названо A и включает в себя идентификаторы i и j, а также функцию Func(). Обратите внимание, что функция Func() полностью определена в границах определения именованного пространства имен. Определение функции можно разместить и за пределами определения именованного пространства имен, но в этом случае в определении функции перед ее именем необходимо в качестве префикса указать имя соответствующего пространства имен, аналогично тому, как это делается при определении функции-члена класса за пределами определения класса. Такая форма определения функции из именованного пространства имен показана в листинге 26.15.

Листинг 26.14. Определение именованного пространства имен

```
namespace A
{
        int i;
        int j;

        int Func()
        {
                return 1;
        }
}
```

Листинг 26.15. Определение функции за пределами именованного пространства имен

```
namespace A
{
        int i;
        int j;

        int Func();
}

int A::Func()
{
                return 1;
}
```

На заметку Именованное пространство имен должно быть определено на уровне видимости файла или внутри другого определения именоввнного пространства имен. Его нельзя определить, скажем, в пределах функции.

Определение области видимости пространства имен

Именованное пространство нмен добавляет новый уровень в иерархню областей виднмостн создаваемой программы, а это означает, что необходнмо нметь некоторый способ идентнфнкацнн области видимостн. Ее идентнфнкатором, безусловно, должно являться имя пространства нмен, которое следует нспользовать в программе для разрешення ссылок на идентнфнкаторы. Напрнмер, для обращення к переменной i нз пространства нмен A нспользуется следующая конструкцня:

```
A::i = 0;
```

Прн необходимостн можно определять вложенное нменованное пространство нмен внутрн определения другого пространства нмен, как показано в лнстннге 26.16. Однако в случаях, подобных показанному в этом листннге, прн разрешеннн ссылок на ндентнфнкаторы необходнмо нспользовать более сложные конструкцнн, обеспечнвающне возможность разлнчать переменные i, объявленные н в пространстве A, н в пространстве B:

```
A::i = 0;
A::B::i = 0;
```

Листинг 26.16. Вложенные определения именованных пространств имен

```
namespace A
{
        int i;
        int j;

        int Func()
        {
                return 1;
        }

        namespace B
        {
                int i;
        }
}
```

Если переменные и функции, определенные в пространстве имен A, используются достаточно часто, то многократного использования конструкции A:: можно избежать, поместив перед работающими с такими переменными и функциями операторами строку using, как показано в листинге 26.17.

Листинг 26.17. Указание используемого пространства имен с помощью оператора using

```
using namespace A;
i = 0;
j = 0;
int num1 = Func();
```

Неименованные пространства имен

Чтобы окончательно сбить вас с толку, язык Visual C++ допускает использование неименованных пространств имен. Неименованное пространство имен определяется так же, как и именованное, за исключением того, что вы опускаете его имя. В листинге 26.18 дано определение неименованного пространства имен. Данный механизм позволяет создавать переменные, имена которых действительны только в данном пространстве имен и к которым невозможно получить доступ откуда-либо еще, поскольку никому неизвестно имя неименованного пространства имен.

Листинг 26.18. Определение неименованного пространства имен

```
namespace
{
        int i;
        int j;

        int Func()
        {
                return 1;
        }
}
```

В пределах неименованного пространства имен обращение к идентификаторам выполняется без указания имени пространства, как показано ниже:

```
i = 0;
j = 0;
int num1 = Func();
```

Псевдонимы именованных пространств имен

Рано или поздно, но вы столкнетесь с пространствами имен, имеющими достаточно длинные имена. В подобных случаях необходимость снова и снова использовать в программе длинное имя пространства имен при обращении к определенным в нем идентификаторам может показаться достаточно утомительной процедурой. Для устранения подобных проблем Visual C++ предоставляет вам возможность определять *псевдонимы именованных пространств имен,* предназначенные просто для подмены имен именованных пространств имен. Псевдоним создается следующим образом:

```
namespace A = LongName;
```

Здесь LongName — это настоящее имя именованного пространства имен, а A — это назначаемый ему псевдоним. После выполнения обработки указанной строки к именованному пространству имен LongName можно обращаться, используя либо имя A, либо имя LongName. Псевдонимы можно рассматривать как прозвище или краткую форму имени. В листинге 26.19 приведен текст небольшой программы, иллюстрирующей работу с псевдонимами именованных пространств имен.

Листинг 26.19. Использование псевдонима именованного пространства имен

```
Namespace ThisIsANamespaceName
{
        int i;
        int j;

        int Func()
        {
                return 2;
        }
}

int main()
{
        namespace ns = ThisIsANamespaceName;

        ns::i = 0;
        ns::j = 0;
        int num1 = ns::Func();

        return 0;
}
```

Обзор новых ключевых слов и типов данных

Согласно рекомендациям комитета ANSI, работающего над Standard C++, в Visual C++ добавлен ряд новых ключевых слов. К ним относятся: bool, true, false, mutable, typename и explicit.

Тип данных bool

Новый тип данных bool является специальным целочисленным типом, который может принимать только значения true и false. Это формальная замена неформальных конструкций #define и typedef, которыми программисты на C++ пользовались многие годы. Большинство условных выражений, таких как (i != 0) или (a<b), теперь возвращают значение bool, а не int.

Ключевое слово mutable

Ключевое слово mutable используется для того, чтобы считать данный объект исключением из правил обработки C-объектов с квалификатором const. К примеру, предположим, что у нас имеется класс Account, в котором необходимо выполнять одни и те же значительные по объему вычисления в начале каждой нз его функций-членов. Вы решили хранить результат этих вычислений в закрытой переменной-члене данного объекта. Подобный класс может выглядеть следующим образом.

```
class Account
{
private:
// Различные данные.
        float value;
        bool valueok;
public:
        void PrintStatement(CTime starttime);
        float GetValue();
        void CreditSalesRep(SalesRep& owner);
        void Deposit (float amt);
        // И так далее.
private
        void UpdateValue();
};

void Account::PrintStatement(CTime starttime)
{
        if (!valueok)
        {
                UpdateValue();
                valueok = true;
        }
        // Концовка функции.
}

void Account::Deposit (float amt)
{
        valueok = false;
        // Dкончание функции.
}
// Остальные функции.
```

Как правило, определение класса приобретает подобный вид не сразу. Вероятно, прежде все функции при каждом обращении просто вызывали `UpdateValue()`, пока кто-то не обратил внимание на низкую производительность программы и не решил хранить значение `value` непосредственно в объекте, чтобы не требовалось вызывать функцию `UpdateValue()` столь часто. Эти изменения затронули только раздел `private`-объекта и не имели никакого влияния на остальной текст программы, использующий этот объект.

Однако тут есть маленькое "но": что если одна из функций-членов была объявлена с квалификатором `const`? Функция `PrintStatement`, например, действительно не изменяет объект `Account` (по крайней мере, она не делала этого до тех пор, пока не была добавлена переменная `valueok`). Но теперь, если вы попробуете объявить функцию `PrintStatement()`, как имеющую квалификатор `const`, компилятор выдаст сообщение об ошибке, поскольку она меняет значение переменной `valueok`. Если оператор `valueok=true` перенести в конец функции `UpdateValue()` и объявить эту функцию-член с квалификатором `const`, то функция `PrintStatement()` скомпилируется успешно, но при компиляции функции `UpdateValue()` на строке `valueok=true` будет сформировано сообщение об ошибке.

Раньше программисты обходили подобные ситуации, вообще отказываясь от использования квалификатора `const`, что было опасно и ухудшало читабельность программы. Теперь же нужно просто объявить переменные `value` и `valueok` с квалификатором `mutable`:

```
private:
        // Различные переменные.
        mutable float value;
        mutable bool valueok;
```

Это объявление указывает на то, что правила работы с квалификатором const ие распространяются на эти переменные-члены, и они могут быть изменены даже функциями, объявленными с квалификатором const. Это дает вам возможность сохранить "концептуальную константность" функций и повысить читабельность создаваемых программ.

Ключевое слово typename

Это ключевое слово применяется только в шаблонах и означает, что используемое вами в данном случае имя является именем типа, а ие именем переменной. Оно также может заменять слово class в определении шаблона.

Ключевое слово explicit

Это ключевое слово уже встречалась нам выше в этой же главе в определении шаблона класса auto_ptr. Оно может использоваться только в конструкторах и присутствует только в их объявлении, как показано ниже.

```
class Foo
{
        explicit Foo() {data=0;} // Допустимо.
        explicit Foo(int i);     // Допустимо.
        // Остальная часть класса.
};
explicit Foo::Foo(int i)         // Так нельзя.
{
        data = i;
}
Foo::Foo(int i) // Допустимо, так как в описании класса для этого
                                 // конструктора указан квалификатор. explicit
{
        data = i;
}
```

Что же означает описание конструктора с квалификатором explicit? Это означает, что для него не должны применяться скрытые преобразования, которые обычно выполняются компиляторами. Рассмотрим следующий фрагмент текста программы:

```
void func(Foo f);
// ...
func(3);
// ...
```

Во время компиляции этого фрагмента компилятор попробует преобразовать тип int в тип Foo. Вероятнее всего, он сформирует программный текст, похожий на следующий:

```
{
Foo compiler_temporary (3);
func(compiler_temporary);
}
```

Если вы добавите в конструктор и деструктор операторы трассировки (см. главу 24), то сможете увидеть, насколько часто компилятору приходится выполнять неявные преобразования подобного типа.

Ключевое слово explicit указывает компилятору на то, что в конструкторе нельзя проводить подобные неявные преобразования. Поскольку конструктор Foo(), принимающий значение тнпа int, объявлен как explicit, компилятор не сможет выполнять такие преобразования и вызов функции func не будет компилироваться.

Спрашивается, зачем нужно пнсать такой программный текст, который невозможно компилировать? Дело в том, что если бы он скомпилировался, то было бы еще хуже. Если бы Foo был управляемым указателем, то когда переменная compiler_temporary вышла бы из областн видимостн, память, на которую она указывала, была бы удалена, а в программе возникла бы серьезная ошибка. Если же вы хотите использовать функцию func, самостоятельно создайте объект тнпа Foo н передайте его в качестве аргумента. Только в таком случае вы избежнте ошибки.

ГЛАВА 27

Многозадачность на основе потоков Windows

В этой главе...

Простая многозадачность на уровне приложения

Взаимодействие между потоками

Синхронизация работы потоков

Вам наверняка известно, что, работая в Windows 95 (или другой современной операционной системе), можно запускать на выполнение несколько программ одновременно. Такая возможность называется многозадачностью на уровне операционной системы. Однако вы можете не знать, что многие новейшие операционные системы также допускают использование *потоков* (thread), представляющих собой отдельные задачи, не являющиеся полноценными приложениями. Поток во многом напоминает подпрограмму. Приложение может создать несколько потоков — несколько различных и выполняющихся параллельно вычислительных процессов — и обеспечить управление их одновременной работой. Потоки позволяют реализовать принцип многозадачности внутри приложения, которое, в свою очередь, также выполняется в режиме многозадачности на уровне операционной системы. Пользователь знает, что у него есть возможность запускать несколько приложений одновременно. Программист знает, что каждое приложение может одновременно запускать несколько потоков. Из этой главы вы узнаете, как создавать потоки в приложении и управлять ими.

Простая многозадачность на уровне приложения

Поток — это ветвь выполнения внутри программы. В многозадачном приложении каждый поток (задача) имеет свой собственный стек и работает независимо от любых других потоков, которые могут быть запущены в этой же программе. При работе с библиотекой MFC различают *UI-потоки* (потоки пользовательского интерфейса), которые работают с сообщениями и обычно решают задачи, связанные с пользовательским интерфейсом, и *рабочие потоки*, которые решают другие задачи.

На заметку Любое приложение всегда имеет, по крайней мере, один поток, который является первичным или главным потоком. Вы можете запускать и останавливать сколько угодно дополнительных потоков, но главный поток будет выполняться до тех пор, пока приложение будет активно.

Поток представляет собой объект самого нижнего уровня в иерархии выполняемых задач. Объектом более верхнего уровня является *процесс*. Как правило, с точки зрения операционной системы любое выполняемое приложение является отдельным процессом. Если некоторое приложение запускается повторно (например, Notepad), в операционной системе формируется новый процесс. Но возможно, что несколько экземпляров одного и того же приложения разделяют один и тот же процесс. Например, если в Internet Explorer вы выберете команду File⇨New Window, то на панели задач у вас будет два приложения, разделяющих один и тот же процесс. Неприятным следствием такого разделения является то, что при крахе одного экземпляра приложения второй также прекратит существование.

Для создания потока с помощью библиотеки MFC достаточно разработать функцию, которая выполняется параллельно с остальной частью приложения. Для запуска потока, в котором эта функция будет выполняться, необходимо вызвать функцию `AfxBeginThread()`. Поток будет оставаться активным до тех пор, пока выполняется запущенная в нем функция. Когда функция закончит свою работу, поток будет уничтожен. Простой вызов функции `AfxBeginThread()` выглядит следующим образом:

```
AfxBeginThread(ProcName, param, priority);
```

В приведенном выше операторе `ProcName` — имя выполняемой в потоке функции, `param` — 32-разрядный параметр, который при необходимости может быть передан потоку, а `priority` определяет приоритет потока и выбирается из списка предопределенных констант. Эти константы и краткое их описание приведены в табл. 27.1.

Таблица 27.1. Константы, определяющие приоритет потока

Константа	Назначение
THREAD_PRIORITY_ABOVE_NORMAL	Устанавливает приоритет на один пункт выше нормального
THREAD_PRIORITY_BELOW_NORMAL	Устанавливает приоритет на один пункт ниже нормвльного
THREAD_PRIORITY_HIGHEST	Устанавливает приоритет на два пункта выше нормвльного
THREAD_PRIORITY_IDLE	Устанавливает базовый приоритет равным 1. Для процесса REALTIME_PRIORITY_CLASS устанавливает приоритет равным 16
THREAD_PRIORITY_LOWEST	Устанавливает приоритет на два пункта ниже нормвльного
THREAD_PRIORITY_NORMAL	Устанавливает нормвльный приоритет
THREAD_PRIORITY_TIME_CRITICAL	Устанавливает базовый приоритет равным 15. Для процесса REALTIME_PRIORITY_CLASS устанавливает приоритет равным 30

На заметку Приоритет потока определяет, как часто по отношению к другим выполняющимся потокам система будет передааать упраление данному потоку. Обычно чем аыше значение приоритета, тем больше аремени аыделяется потоку на аыполнение. Вот почему значение THREAD_PRIORITY_TIME_CRITICAL так валико.

Для того чтобы на реальном примере увидеть работу простого потока, создайте приложение Thread. С этой целью выполните следующие операции.

1. Запустите AppWizard и укажите ему на необходимость создания нового проекта класса MFC AppWizard (exe) с именем Thread, как показано на рис. 27.1.
2. Задайте для нового проекта параметры настройки AppWizard, приведенные ниже. По завершении ввода установок диалоговое окно New Project Information должно выглядеть так, как показано на рис. 27.2.

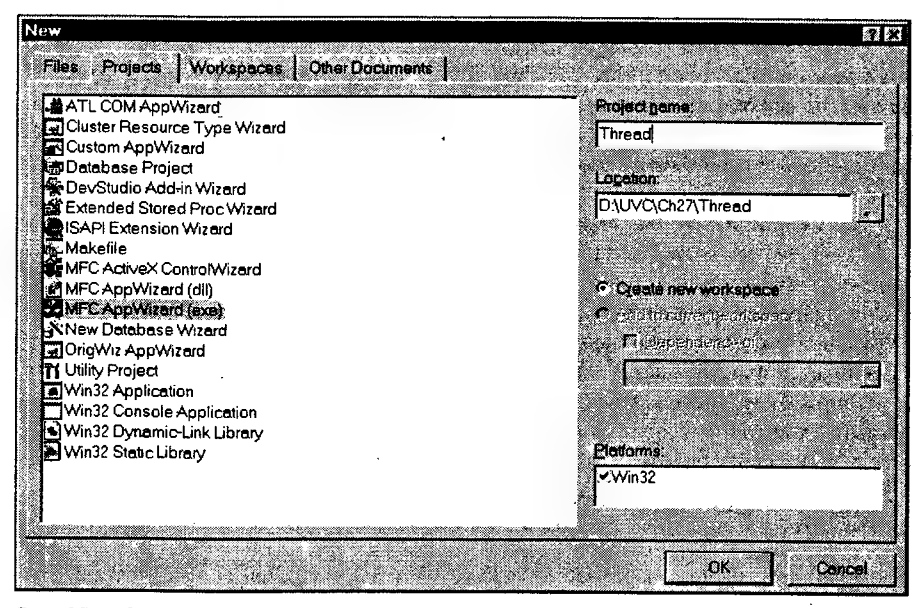


Рис. 27.1. Создание с помощью AppWizard нового проекта с именем Thread

Этап 1. Установите опцию Single document.

Этап 2. Оставьте настройку по умолчанию.

Этап 3. Оставьте настройку по умолчанию.

Этап 4. Сбросьте все флажки.

Этап 5. Оставьте настройку по умолчанию.

Этап 6. Оставьте настройку по умолчанию.

2. Используя редактор ресурсов, добавьте в меню приложения IDR_MAINFRAME новое меню Thread. Поместите в него одну команду с названием Start Thread (Запустить поток) и идентификатором ID_STARTTHREAD, введите текст контекстной подсказки, как показано на рис. 27.3.
3. С помощью ClassWizard свяжите команду ID_STARTTHREAD с функцией обработки сообщения OnStartthread(), как показано на рис. 27.4. Перед добавлением этой функции убедитесь, что в поле Class Name выбрано значение CThreadView.
4. Щелкните на кнопке Edit Code и введите приведенные ниже операторы в новую функцию OnStartthread(), заменив ими комментарий // TODO: Add your command handler code here.

```
HWND hWnd = GetSafeHwnd();
AfxBeginThread(ThreadProc, hWnd, THREAD_PRIORITY_NORMAL);
```

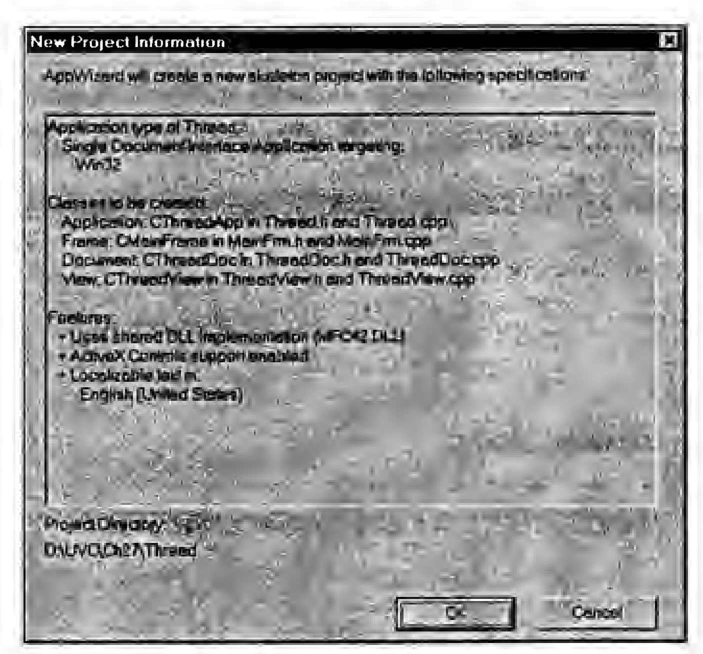


Рис. 27.2. Настройка AppWizard для проекта Thread

Рис. 27.3. Добавление меню Thread, *содержащего одну команду* Start Thread

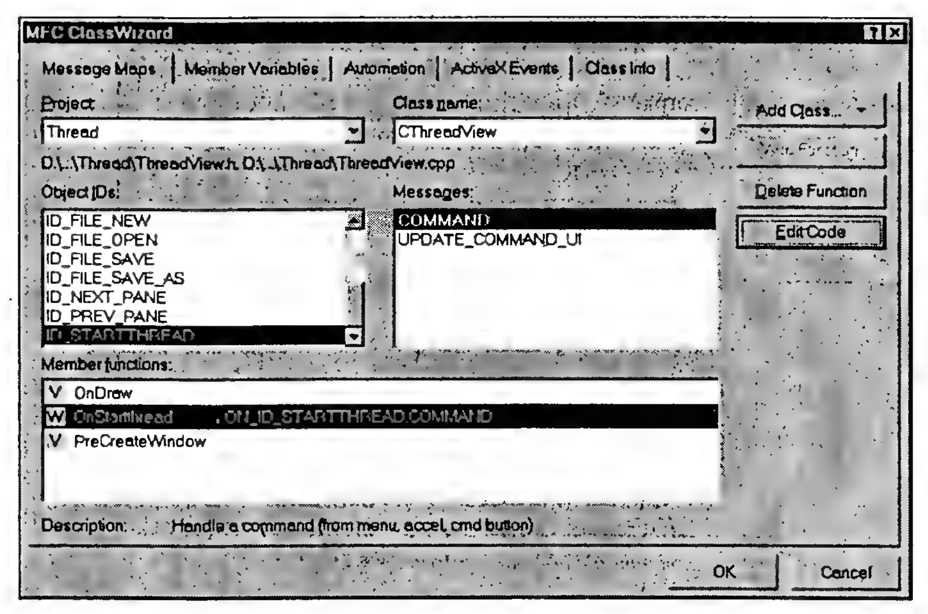


Рис. 27.4. Добавление в класс `CThreadView` *функции обработки сообщения* `OnStartthread()`

В этом фрагменте текста программы вызывается функция ThreadProc(), которая будет работать в своем собственном потоке. Далее в файл ThreadView.cpp добавьте функцию ThreadProc(), текст которой представлен в листинге 27.1, поместив ее непосредственно перед функцией OnStartthread(). Обратите внимание, что функция ThreadProc() является глобальной функцией, а не методом класса CThreadView, несмотря на то, что она находится в файле, в котором реализован этот класс.

Листинг 27.1. Фрагмент файла ThreadView.cpp — функция ThreadProc()

```
UINT ThreadProc(LPVOID param)
{
        ::MessageBox((HWND)param, "Thread activated.", "Thread", MB_OK);

        return 0;
}
```

Эта функция потока просто сообщает о начале своей работы. Функция SDK MessageBox() очень похожа на функцию AfxMessageBox(). Но поскольку функция ThreadProc() не является функцией-членом класса, производного от CWnd, использовать в ней функцию AfxMessageBox() невозможно.



Два двоеточия перед именем функции говорят о том, что вызывается глобальная функция, а не функция-член класса из библиотеки MFC. Для программистов, создающих приложения Windows, это обычно означает вызов функции API или SDK. Например, внутри класса окна библиотеки MFC для вывода пользователю сообщения "Hi, There!" можно вызвать функцию MessageBox("Hi, there!"). Такая форма вызова функции MessageBox() указывает на то, что она является функцией-членом класса окна MFC. Для вызова оригинальной Windows-версии этой функции следует использовать оператор ::MessageBox(0, "Hi, There!", "Message", MB_OK). Обратите внимание на два двоеточия перед именем функции и дополнительные аргументы.

Когда вы запустите программу Thread, на экране раскроется ее главное окно. Выберите команду Thread⇨Start Thread, и система запустит поток, представленный функцией ThreadProc(). Начав работу, эта функция выведет окно сообщения, показанное на рис. 27.5.

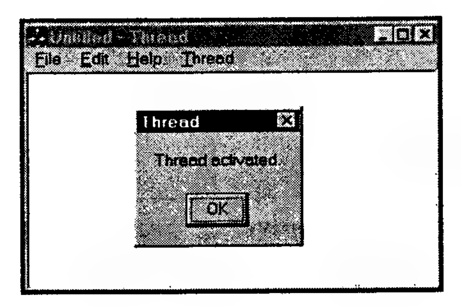


Рис. 27.5. Простой вторичный поток в программе Thread выводит окно сообщений, после чего заканчивает свою работу

Взаимодействие между потоками

Как правило, вторичный поток решает некоторую задачу для главной программы, что предполагает обязательное существование некоторого канала связи между программой (которая также является потоком) и порожденным ею вторичным потоком. Существует несколько способов решения подобных задач взаимодействия — использование глобальных переменных, использование объектов событий и использование сообщений. В данном разделе мы рассмотрим все эти способы взаимодействия потоков.

Взаимодействие потоков с помощью глобальных переменных

Предположим, что в главной программе необходимо остановить работу потока. Значит, требуется найти способ сообщить потоку о том, что ему следует завершить работу. Один из способов решения этой задачи заключается во введении глобальной переменной, которая будет анализироваться в потоке, чтобы обнаружить значение, указывающее на необходимость завершить работу. Для проверки работы этого механизма на практике модифицируем приложение Thread.

1. С помощью редактора ресурсов добавьте команду Stop Thread в меню Thread приложения. Присвойте этой новой команде идентификатор ID_STOPTHREAD, как показано на рис. 27.6.

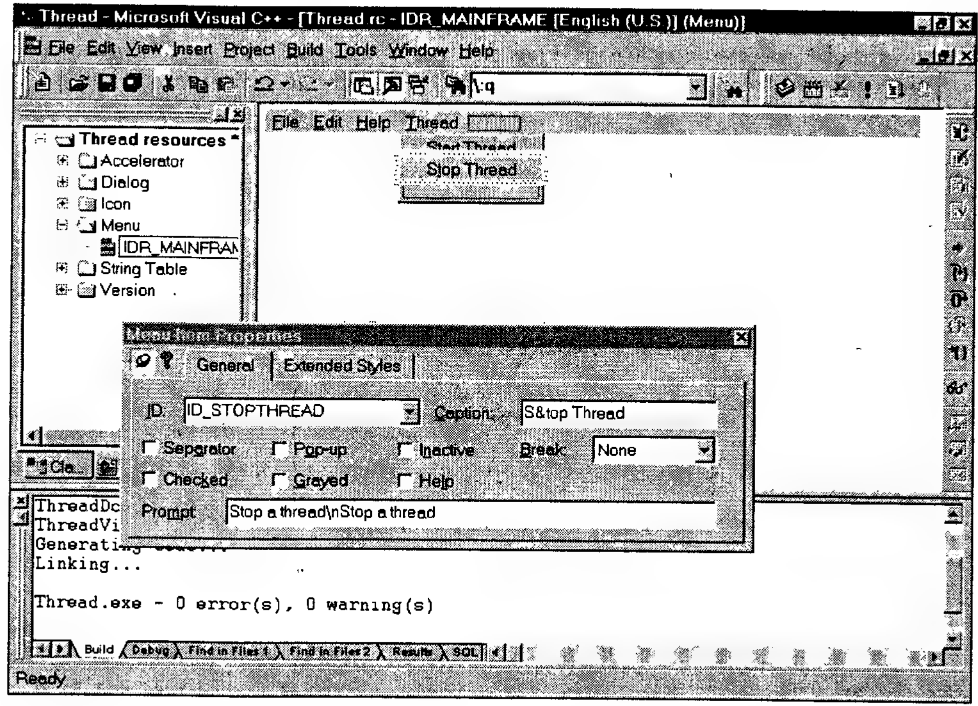


Рис. 27.6. Добавление в меню Thread команды Stop Thread

2. С помощью ClassWizard свяжите команду ID_STOPTHREAD с функцией обработки сообщения OnStopthread(), как показано на рис. 27.7. Перед тем как добавить новую функцию, убедитесь, что в поле **Class Name** выбрано значение CThreadView. Вместо комментария

```
// TODO: Add your command handler code here
```

включите в функцию OnStopthread()следующий оператор:

```
threadController = 0;
```

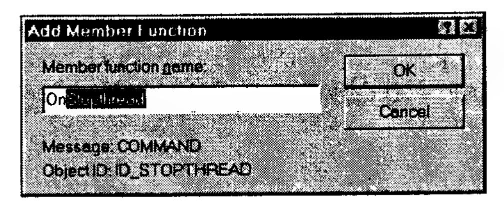


Рис. 27.7. Добавление функции обработки сообщения OnStopthread()

В этом операторе производится обращение к новой глобальной переменной, которую нам сейчас предстоит объявить.

3. Включите в начало файла ThreadView.cpp, сразу же после директивы #endif, следующую строку:

```
volatile int threadController;
```

Ключевое слово volatile означает, что вы санкционируете изменение этой переменной вне использующего ее потока. Это зарезервированное слово требует от компилятора не размещать данную переменную для хранения в регистре или каким-либо иным способом, препятствующим внешнему изменению значения этой переменной. Изменение значения этой переменной может происходить даже в том случае, если в процессе выполнения данного потока не встречались операторы, модифицирующие ее значение.

4. В функцию OnStartthread() перед двумя внесенными в нее ранее строками добавьте следующий оператор:

```
threadController = 1;
```

Вы, вероятно, уже догадались, что значение переменной threadController будет указывать потоку, следует ли ему продолжать работу. Замените существующий текст функции ThreadProc() текстом, приведенным в листинге 27.2.

Листинг 27.2. Новый вариант функции ThreadProc()

```
UINT ThreadProc(LPVOID param)
{
        ::MessageBox((HWND)param, "Thread Activated", "Thread", MB_OK);

        while (threadController == 1)
        {
                ;
        }

        ::MessageBox((HWND)param, "Thread Stopped", "Thread", MB_OK);

        return 0;
}
```

Теперь функция потока в начале работы выведет окно сообщения, информирующее пользователя о запуске потока. Затем будет выполняться цикл while, проверяющий значение глобальной переменной threadController и ожидающий изменения ее значения на 0. Хотя данный цикл while совершенно тривиален, именно в нем следует поместить программный текст той задачи, которая должна решаться в потоке. Однако этот текст должен быть построен так, чтобы очередная проверка значения переменной threadController не откладывалась на слишком долгое время.

Откомпилируйте программу и запустите ее на выполнение. Для запуска вторичного потока выберите команду Thread⇒Start Thread. Должно появиться окно сообщений, информирующее о том, что второй поток запущен. Остановите вторичный поток, выбрав команду Thread⇒Stop Thread. Вновь появится окно сообщений, на этот раз информирующее о том, что вторичный поток завершил свою работу.

Внимание!

Использование глобальных переменных для организации взаимодействия между потоками является, грубо говоря, весьма примитивным подходом к организации подобных связей. Он даже может быть опасным, если вы не очень хорошо представляете на уровне языка ассемблера, как в C++ реализуется работа с переменными. Другие технологии организации взаимодействия между потоками более надежны и элегантны.

Взаимодействие потоков с помощью сообщений

Теперь вы знакомы с простым, хотя и довольно примитивным методом передачи информации потоку от главной программы. Ну, а наоборот? Как из потока передать информацию основной программе? Простейший способ состоит во включении в программу определяемых пользователем сообщений Windows.

На первом шаге следует определить пользовательское сообщение, что легко реализуется следующим оператором:

```
const WM_USERMSG = WM_USER + 100;
```

Константа WM_USER, определенная в Windows, хранит первый из доступных номеров пользовательских сообщений. Поскольку другие части программы могут применять пользовательские сообщения для своих собственных целей, в приведенной выше строке константе WM_USERMSG присваивается значение WM_USER + 100.

После определения сообщение может быть в любой момент передано из потока главной программе посредством вызова функции ::PostMessage(). (Обработка сообщений обсуждалась в главе 3. Передача собственного сообщения позволяет воспользоваться преимуществами встроенных в MFC функций обработки сообщений.) Стандартный вызов функции ::PostMessage() выглядит следующим образом:

```
::PostMessage((HWND)param, WM_USERMSG, 0, 0);
```

Функция PostMessage() включает четыре аргумента: дескриптор окна, в которое посылается сообщение, идентификатор сообщения и параметры сообщения WPARAM и LPARAM.

Для того чтобы на практике увидеть работу метода передачи пользовательских сообщений из потока, модифицируйте приложение Thread так, как описано ниже.

1. Добавьте в начало файла заголовка ThreadView.h, непосредственно перед объявлением класса, следующую строку:

   ```
   const WM_THREADENDED = WM_USER + 100;
   ```

2. В этом же файле добавьте в карту сообщений приведенную ниже строку, поместив ее сразу после комментария AFX_MSG и перед строкой DECLARE_MESSAGE_MAP:

```
afx_msg LONG OnThreadended(WPARAM wParam, LPARAM lParam);
```

3. Откройте файл ThreadView.cpp и добавьте приведенную ниже строку в карту сообщений класса, обязательно поместив ее непосредственно *после* комментария }}AFX_MSG_MAP:

```
ON_MESSAGE(WM_THREADENDED, OnThreadended)
```

4. Замените существующий текст функции ThreadProc() новым вариантом, приведенным в листинге 27.3.

Листинг 27.3. Функция ThreadProc(), реализующая посылку сообщения

```
UINT ThreadProc(LPVOID param)
{
        ::MessageBox((HWND)param, "Thread Activated", "Thread", MB_OK);

        while (threadController == 1)
        {
                ;
        }

        ::PostMessage((HWND)param, WM_THREADENDED, 0, 0);

        return 0;
}
```

5. Добавьте в конец файла ThreadView.cpp функцию, текст которой приведен в листинге 27.4.

Листинг 27.4. Файл ThreadView.cpp — функция CThreadView::OnThreadended()

```
LONG CThreadView::OnThreadended(WPARAM wParam, LPARAM lParam)
{
        AfxMessageBox("Thread ended.");
        return 0;
}
```

Откомпилируйте приложение и запустите его на выполнение. Для запуска вторичного потока выберите команду Thread⇨Start Thread. Появится окно сообщений, информирующее о запуске потока. Для прекращения работы потока выберите команду Thread⇨Stop Thread. Как и в предыдущей версии программы, появится окно сообщений, информирующее о завершении работы потока.

Хотя может показаться, что эта версия приложения Thread работает идентично предыдущей, в новой версии есть одно существенное отличие. Теперь приложение отображает окно сообщения, информирующее о завершении работы потока, из главной программы, а не из самого потока. Происходит это по той причине, что при выборе пользователем команды Stop Thread поток посылает главной программе сообщение WM_THREADENDED. Получив это сообщение, главная программа выводит окно сообщений с фразой Thread stopped.

Взаимодействие потоков с помощью объектов событий

Несколько более сложным методом организации взаимодействия между потоками является использование *объектов событий*, представленных в библиотеке MFC классом CEvent. Объект события может находиться в одном из двух состояний — *сигнализирует* или *молчит*.

Потоки отслеживают момент, когда объект события начнет сигнализировать, и своевременно выполняют требуемые от них операции. Создать объект события не сложнее, чем объявить глобальную переменную, например:

```
CEvent threadStart;
```

Хотя конструктор объекта CEvent имеет множество необязательных аргументов, их обычно опускают, создавая объект по умолчанию, как показано в приведенной выше строке. При создании объект события автоматически устанавливается в состояние *молчит*. Для перевода объекта события в состояние *сигнализирует* необходимо вызвать метод SetEvent() этого объекта , как показано ниже:

```
threadStart.SetEvent();
```

После выполнения такого оператора объект события threadStart будет переведен в состояние *сигнализирует*. Поток должен следить за моментом изменения состояния и таким образом определить, когда следует приступить к работе. Но как поток осуществляет подобное слежение? С помощью вызова функции API Windows WaitForSingleObject():

```
::WaitForSingleObject(threadStart.m_hObject, INFINITE);
```

Двумя аргументами этой функции являются:

- дескриптор отслеживаемого события (хранится в переменной-члене объекта события m_hObject);
- время, в течение которого функция будет ожидать это событие.

Предопределенная константа INFINITE требует от функции WaitForSingleObject() не возвращать управление, пока указанное событие не будет переведено в состояние *сигнализирует*. Другими словами, если поместить приведенную выше строку в начало функции потока, система приостановит выполнение потока до тех пор, пока указанный объект события не будет отмечен. Несмотря на то что поток уже создан и начал работать, он будет остановлен вплоть до момента, когда произойдет то событие, которого он ожидает. Когда программа будет готова к тому, чтобы поток приступил к решению возложенной на него задачи, в ней следует вызвать функцию SetEvent(), как об этом упоминалось выше.

Как только приостановка работы потока будет отменена, он продолжит выполнение порученного ему задания. Но теперь, если главная программа должна указать потоку на необходимость завершить работу, поток должен отслеживать изменение состояния другого объекта события. Чтобы опросить событие, необходимо вновь вызвать функцию WaitForSingleObject(), но на этот раз указать время ожидания равным нулю, как показано в следующем примере:

```
::WaitForSingleObject(threadEnd.m_hObject, 0);
```

Если вызванная подобным образом функция WaitForSingleObject() возвратит значение WAIT_OBJECT_0, значит, объект события находится в состоянии *сигнализирует*. В противном случае состояние объекта события — *молчит*.

Чтобы на примере опробовать работу с объектами событий, выполните следующие операции по дальнейшей модификации приложения Thread.

1. Добавьте в начало файла ThreadView.cpp приведенную ниже строку, поместив ее сразу после строки #include "ThreadView.h":

   ```
   #include "afxmt.h"
   ```

2. Добавьте в начало файла ThreadView.cpp приведенные ниже строки, поместив их после недавно добавленной вами строки volatile int threadController:

```
CEvent threadStart;
CEvent threadEnd;
```

3. Удалите из файла строку

```
volatile int threadController;
```

4. Замените имеющийся текст функции ThreadProc() новым вариантом, приведенным в листинге 27.5.

Листинг 27.5. Очередная версия функции ThreadProc()

```
UINT ThreadProc(LPVOID param)
{
	::WaitForSingleObject(threadStart.m_hObject, INFINITE);
	::MessageBox((HWND)param, "Thread activated.", "Thread", MB_OK);

	BOOL keepRunning = TRUE;
	while (keepRunning)
	{
		int result = ::WaitForSingleObject(threadEnd.m_hObject, 0);
		if (result == WAIT_OBJECT_0)
			keepRunning = FALSE;
	}

	::PostMessage((HWND)param, WM_THREADENDED, 0, 0);

	return 0;
}
```

5. Удалите тело функции OnStartthread() и вставьте вместо него следующую строку:

```
threadStart.SetEvent();
```

6. Замените весь текст функции OnStopthread() следующей единственной строкой:

```
threadEnd.SetEvent();
```

7. С помощью ClassWizard добавьте функцию OnCreate() (рис. 27.8), которая будет обрабатывать сообщение WM_CREATE. Перед добавлением этой функции убедитесь, что в списке **Class Name** выбрано значение CThreadView.

8. Добавьте в функцию OnCreate() приведенные ниже строки, заменив ими комментарий // TODO: Add your specialized creation code here:

```
HWND hWnd = GetSafeHwnd();
AfxBeginThread(ThreadProc, hWnd);
```

И вновь, на первый взгляд, работа новой версии программы не отличается от работы ее предыдущей версии. Тем не менее теперь для организации взаимодействия между главной программой и потоком используются как объекты событий, так и определенные пользователем сообщения Windows. В использовании глобальных переменных больше иет нужды.

Еще одио существенное различие между этой и предыдущей версиями программы заключается в том, что вторичный поток запускается функцией OnCreate(), вызываемой один раз после запуска приложеиия и формирования его представления. Однако, поскольку в первой строке потоковой функции находится вызов WaitForSingleObject(), поток сразу же приостанавливает свое выполнение и ожидает переключения состояния объекта события threadStart.

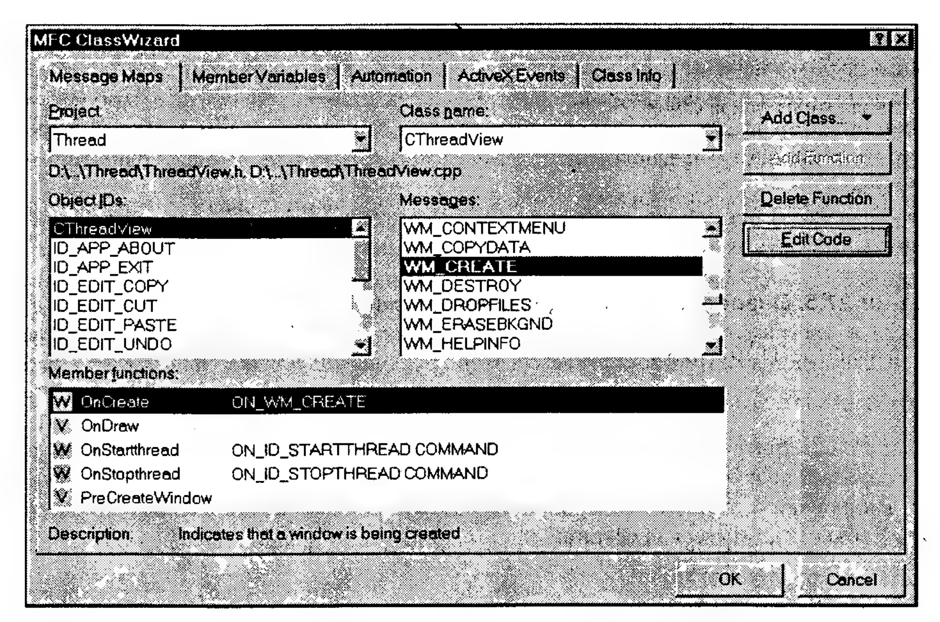


Рис. 27.8. Добавление функции `OnCreate()` *с помощью* `ClassWizard`

Когда объект события `threadStart` будет установлен в состояние *сигнализирует*, выполнение потоковой функции возобновляется и она выводит окно сообщения. Далее поток входит в цикл `while`, в котором выполняется опрос объекта события `threadEnd`. Цикл `while` будет продолжаться до тех пор, пока объект `threadEnd` не будет установлен в состояние *сигнализирует*, после чего функция потока отправляет главной программе сообщение `WM_THREADENDED` и завершает свою работу. Поток запускается в функции `OnCreate()`, поэтому повторный его запуск невозможен.

Синхронизация работы потоков

Использование в программе нескольких потоков одновременно может привести к возникновению ряда специфических проблем. Например, как предотвратить одновременный доступ двух потоков к одним и тем же данным? Что произойдет, если в тот момент, когда один поток еще не завершил процедуру обновления некоторых данных, другой поток предпринимает попытку эти данные считать? Почти наверняка данные, считанные вторым потоком, окажутся некорректными, поскольку лишь некоторая их часть была на данный момент обновлена.

Обеспечение корректной совместной работы потоков называется *синхронизацией потоков*. Рассмотренные нами объекты событий являются одной из форм синхронизации потоков. В этом разделе вы узнаете о *критических секциях*, *защелках* и *семафорах* — объектах синхронизации потоков, которые обеспечивают надежную работу многозадачных приложений.

Использование критических секций

Использование критических секций — это простой способ обеспечить доступ к набору данных только одного из потоков. Создавая критическую секцию, вы передаете потокам объект, который они должны использовать совместно. Любой поток, владеющий объектом критической секции, получает доступ к защищенным данным. Остальные потоки вынуждены ожидать освобождения критической секции, захваченной первым потоком, и только после

этого какой-либо из них сможет захватить данную критическую секцию и, в свою очередь, получить доступ к данным.

Поскольку защищенные данные представляются единственным объектом критической секции и при этом только один поток может владеть данным объектом в каждый конкретный момент времени, доступ к защищенным данным может получить не более одного потока одновременно.

Для создания в программе, использующей библиотеку MFC, объекта критической секции необходимо создать экземпляр объекта класса `CCriticalSection`, как это показано ниже:

```
CCriticalSection criticalSection;
```

Когда в программе необходимо получить доступ к данным, защищенным критической секцией, вызывается метод `Lock()` объекта этой критической секции, как показано ниже:

```
criticalSection.Lock();
```

Если объект критической секции в данный момент не захвачен другим потоком, функция `Lock()` передаст этот объект во владение данному потоку. Теперь поток может получить доступ к защищенным данным. Завершив обработку данных, поток должен вызвать метод `Unlock()` объекта критической секции:

```
criticalSection.Unlock();
```

Функция `Unlock()` освобождает объект критической секции. В результате другой поток сможет его захватить и получить доступ к защищенным данным.

Лучшим способом реализации механизма защиты, использующего критическую секцию, является размещение защищаемых данных в классе защиты потока. Если это сделать, то основной программе больше не придется беспокоиться о синхронизации работы потоков — методы этого класса возьмут все на себя. В качестве примера проанализируйте листинг 27.6, в котором приведен текст файла заголовка для класса массива с защитой потока.

Листинг 27.6. Файл `COUNTARRAY.H` — файл заголовка класса `CCountArray`

```
#include "afxmt.h"

class CCountArray
{
private:
        int array[10];
        CCriticalSection criticalSection;

public:
        CCountArray() {};
        ~CCountArray() {};

        void SetArray(int value);
        void GetArray(int dstArray[10]);
};
```

В начале файла к программе подключается файл заголовка библиотеки MFC `afxmt.h`, обеспечивающий доступ к классу `CCriticalSection`. В объявлении класса `CCountArray` выполняется объявление целочисленного массива из десяти элементов, предназначенного для хранения защищаемых критической секцией данных, а также объявляется объект критической секции `criticalSection`. Открытые методы класса `CCountArray` включают обыкновенный конструктор и деструктор, а также две функции для чтения и записи массива. Именно два последних метода класса и должны работать с объектом критической секции, так как только они имеют доступ к массиву.

В листинге 27.7 приведен текст файла, реализующего класс CCountArray. Обратите внимание, что каждый метод этого класса обеспечивает захват и освобождение объекта критической секции. Это означает, что любой поток может вызвать эти методы, абсолютно не заботясь о синхронизации потоков. Например, если поток номер один вызовет функцию SetArray(), то первое, что сделает эта функция, — будет вызов criticalSection.Lock(), которая передаст объект критической секции во владение этому потоку. Затем весь цикл for выполняется в полной увереииости, что его работа не будет прервана другим потоком. Если в это время поток номер два вызовет функцию SetArray() или GetArray(), то очередной вызов criticalSection.Lock() приостановит работу потока до тех пор, пока поток номер один не освободит объект критической секции. А это произойдет тогда, когда функция SetArray() закончит выполнение цикла for и вызовет criticalSection.Unlock(). Затем система возобновит работу потока номер два, передав ему во владение объект критической секции. Таким образом, все потоки будут терпеливо ожидать своей очереди на получение доступа к защищенным данным.

Листинг 27.7. Файл COUNTARRAY.CPP — реализация класса CCountArray

```
#include "stdafx.h"
#include "CountArray.h"

void CCountArray::SetArray(int value)
{
        criticalSection.Lock();

        for (int x=0; x<10; ++x)
                array[x] = value;

        criticalSection.Unlock();
}

void CCountArray::GetArray(int dstArray[10])
{
        criticalSection.Lock();

        for (int x=0; x<10; ++x)
                dstArray[x] = array[x];

        criticalSection.Unlock();
}
```

Теперь, когда вы уже знаете, что представляет собой класс потока, самое время опробовать его на практике. Выполните перечисленные ниже операции по модификации приложения Thread, что даст возможность с его помощью проверить функционирование класса CCountArray.

1. Выберите команду File⇨New и добавьте в проект новый файл заголовка CountArray.h, как показано на рис. 27.9. Введите в этот файл текст, приведенный в листинге 27.6.
2. Еще раз выберите команду File⇨New и добавьте в проект исходный файл CountArray.cpp. Поместите в него текст, представленный в листинге 27.7.
3. Откройте файл ThreadView.cpp и добавьте в него приведенную ниже строку, поместив ее сразу после строки #include "afxmt.h" (ее мы добавили в этот файл во время предыдущей модификации):

   ```
   #include "CountArray.h"
   ```

4. Добавьте приведенную ниже строку в начало этого же файла, сразу после строки CEvent threadEnd, добавленной в предыдущий раз:

   ```
   CCountArray countArray;
   ```

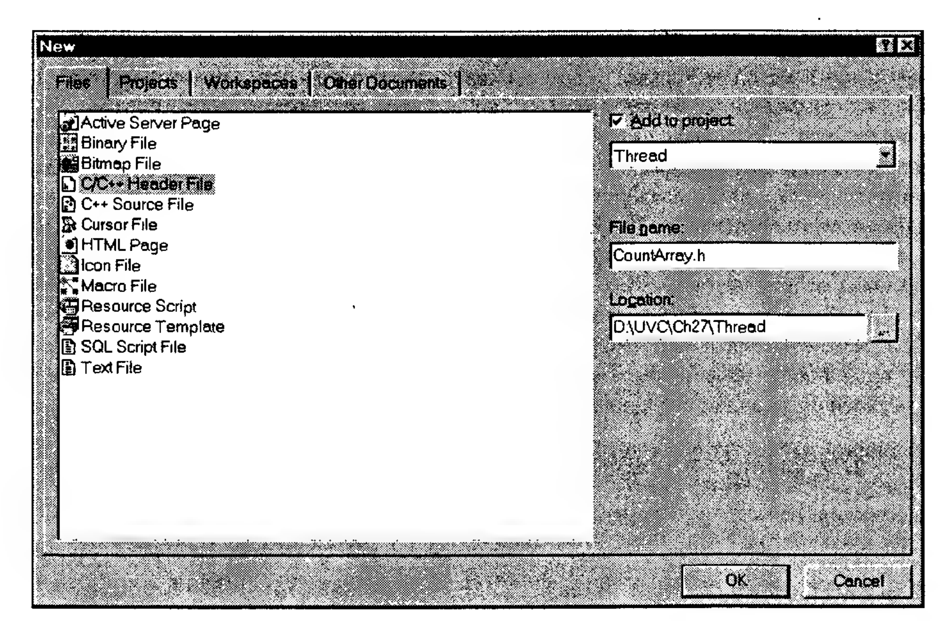


Рис. 27.9. Добавление в проект `Thread` *файла* `CountArray.h`

5. Удалите из файла `ThreadView.cpp` строки

```
CEvent threadStart;
```

и

```
CEvent threadEnd;
```

6. Удалите из карты сообщений строки

```
ON_MESSAGE(WM_THREADENDED, OnThreadended)
ON_COMMAND(ID_STOPTHRHEAD, OnStopthread)
```

и

```
ON_WM_CREATE()
```

7. Замените функцию `ThreadProc()` функциями, приведенными в листинге 27.8.

Листинг 27.8. Функции `WriteThreadProc()` и `ReadThreadProc()`

```
UINT WriteThreadProc(LPVOID param)
{
        for(int x=0; x<10; ++x)
        {
                countArray.SetArray(x);
                ::Sleep(1000);
        }

        return 0;
}

UINT ReadThreadProc(LPVOID param)
{
        int array[10];
```

```
	for (int x=0; x<20; ++x)
	{
		countArray.GetArray(array);
		char str[50];
		str[0] = 0;
		for (int i=0; i<10; ++i)
		{
			int len = strlen(str);
			wsprintf(&str[len], "%d ", array[i]);
		}
		::MessageBox((HWND)param, str, "Read Thread", MB_OK);
	}

	return 0;
}
```

8. Замените весь текст функции `OnStartthread()` следующими операторами:

```
HWND hWnd = GetSafeHwnd();
AfxBeginThread(WriteThreadProc, hWnd);
AfxBeginThread(ReadThreadProc, hWnd);
```

9. Удалите функции `OnStopthread()`, `OnThreadended()` и `OnCreate()` из файла.

10. Откройте файл `ThreadView.h` и удалите строку

```
const WM_THREADENDED = WM_USER+100;
```

11. В этом же файле удалите из карты сообщений строки

```
afx_msg LONG OnThreadended(WPARAM wParam, LPRAM lParam);
afx_msg void OnStopthread();
afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
```

12. Используя редактор ресурсов, удалите команду **Stop Thread** из меню **Thread**.

Откомпилируйте новую версию приложения Thread и запустите ее на выполнение. На экране раскроется главное окно приложения. Для запуска процесса выберите команду **Thread⇨Start Thread**. Первым появится окно сообщений (рис. 27.10), отображающее текущие значения элементов защищенного массива. Каждый раз при закрытии оно будет появляться вновь, отображая обновленное содержимое массива. Всего вывод окна будет повторяться 20 раз. Значения, отображаемые в окне сообщений, будут зависеть от того, насколько быстро вы будете закрывать это окно сообщений. Первый поток записывает новые значения в массив ежесекундно, причем даже тогда, когда вы просматриваете содержимое массива с помощью второго потока.

Обратите внимание на одну важную деталь: второй поток ни разу не прервал работу первого потока во время изменения им значений в массиве. На это указывает идентичность всех десяти значений элементов массива. Если бы работа первого потока прерывалась во время модификации массива, то десять значений массива были бы неодинаковы, как показано на рис. 27.11.

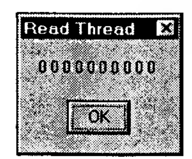


Рис. 27.10. Окно сообщений отображает текущие значения элементов защищенного массива

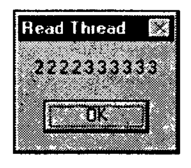


Рис. 27.11. Содержимое окна сообщений при отсутствии синхронизации потоков

Если вы внимательно проанализируете исходный текст программы, то увидите, что первый поток с именем WriteThreadProc() вызывает функцию-член SetArray() класса CCountArray десять раз за один цикл for. В каждом цикле функция SetArray() передает потоку объект критической секции, заменяет содержимое массива переданным ей числом и вновь освобождает объект критической секции. Обратите внимание на вызов функции Sleep(), которая приостанавливает работу потока на число миллисекунд, передаваемых ей в качестве аргумента.

Второй поток ReadThreadProc() также пытается получить доступ к этому объекту критической секции, чтобы иметь возможность сформировать строку на экране, содержащую текущие значения элементов массива. Но, если в данный момент поток WriteThreadProc() предпримет попытку заполнить массив новыми значениями, поток ReadThreadProc() вынужден будет ждать. И наоборот — поток WriteThreadProc() не сможет получить доступ к защищенным данным до тех пор, пока вновь не получит возможность работать с объектом критической секции, освобожденным потоком ReadThreadProc().

Если вы наяву хотите убедиться в том, что объект критической секции работает именно так, как описано выше, удалите строку criticalSection.Unlock(), расположенную в конце метода SetArray() класса CCountArray. Затем откомпилируйте и выполните программу. На этот раз после запуска потоков вы не увидите никаких сообщений. Почему? Потому что поток WriteThreadProc() захватывает объект критической секции и не освобождает его, что заставляет систему остановить работу потока ReadThreadProc() раз и навсегда (или, по крайней мере, до окончания работы программы).

Использование защелок

Защелки (mutex) во многом похожи на критические секции, но являются несколько более сложными объектами, так как обеспечивают безопасное разделение ресурсов не только потоками одного приложения, но также потоками различных приложений. Хотя подробный анализ синхронизации работы потоков нескольких приложений выходит за рамки этой главы, вы можете опробовать работу с защелками, используя их вместо критических секций.

В листинге 27.9 приведен текст файла заголовка для класса CountArray2. За исключением нового имени класса и объекта защелки, этот файл идентичен предыдущей версии файла CountArray.h. В листинге 27.10 приведен текст исходного файла, реализующего этот модифицированный класс. Как видите, его методы, работающие с защелкой вместо критической секции, значительно изменились, хотя оба типа объектов обеспечивают решение, в сущности, совершенно одинаковых задач.

Листинг 27.9. Файл CountArray2.h — файл заголовка класса CCountArray2

```
#include "afxmt.h"

class CCountArray2
{
private:
        int array[10];
        CMutex mutex;

public:
        CCountArray2() {};
        ~CCountArray2() {};

        void SetArray(int value);
        void GetArray(int dstArray[10]);
};
```

Листинг 27.10. Файл CountArray2.cpp — файл реализации класса CCountArray2

```
#include "stdafx.h"
#include "CountArray2.h"

void CCountArray2::SetArray(int value)
{
        CSingleLock singleLock(&mutex);
        SingleLock.Lock();

        for (int x=0; x<10; ++x)
        array[x] = value;
}

void CCountArray2::GetArray(int dstArray[10])
{
        CSingleLock singleLock(&mutex);
        SingleLock.Lock();

        for (int x=0; x<10; ++x)
              dstArray[x] = array[x];
}
```

Для получения доступа к объекту защелки необходимо создать либо объект класса CSingleLock, либо объект класса CMultiLock, который и будет фактически управлять процессом доступа. В классе CCountArray2 используются объекты класса CSingleLock, так как этот класс работает с единственной защелкой. Для работы с защищенными ресурсами (в данном случае — с массивом) в программе необходимо создать объект класса CSingleLock, как показано ниже:

```
CSingleLock singleLock(&mutex);
```

Аргумент конструктора является указателем на обеспечивающий синхронизацию потоков объект, с помощью которого и осуществляется управление. Затем для получения доступа к защелке вызывается метод Lock() объекта класса CSingleLock:

```
singleLock.Lock();
```

Если защелка еще не захвачена, то вызывающий поток становится ее владельцем. Если владельцем защелки уже является другой поток, система приостанавливает работу вызывающего потока до тех пор, пока защелка не будет освобождена, после чего ожидающий поток сможет получить ее в свое распоряжение и продолжить работу.

Для освобождения защелки необходимо вызвать метод Unlock() объекта класса CSingleLock. Однако если вы создадите экземпляр класса CSingleLock в стеке (а не в куче с помощью оператора new), как это показано в листинге 27.10, то вызывать Unlock() вообще нет необходимости. Когда функция SetArray() завершит свою работу, объект выйдет из области видимости, что приведет к вызову его деструктора, который автоматически освободит объект.

Чтобы опробовать, как новый класс CCountArray2 будет работать в приложении Thread, добавьте в проект новые файлы CountArray2.cpp и CountArray2.h, а файлы CountArray.h и CountArray.cpp удалите. В файле ThreadView.cpp замените все обращения к CCountArray обращениями к CCountArray2. Поскольку вся синхронизация потоков организована в классе CCountArray2, никаких дополнительных изменений с целью обеспечения использования защелки вместо критической секции не потребуется. Удобно, не так ли?

Использование семафоров

Хотя семафоры в программах, созданных на основе библиотеки MFC, используются аналогично критическим секциям и защелкам, они выполняют другие функции. Вместо того чтобы в каждый момент времени предоставлять право доступа к ресурсу только одному потоку, семафоры позволяют иметь доступ к ресурсу сразу нескольким потокам, но до определенного предела. Другими словами, семафоры обеспечивают одновременный доступ к ресурсу некоторому количеству потоков, не превышающему заданного максимального значения.

Когда создается семафор, ему сообщается максимальное количество потоков, которым одновременно будет разрешен доступ к ресурсу. Затем всякий раз, когда очередной поток захватывает ресурс, семафор уменьшает значение своего внутреннего счетчика. Когда это значение достигает нуля, больше ни один поток не может получить доступ к ресурсу до тех пор, пока какой-либо поток не освободит его, увеличивая тем самым значение счетчика семафора.

Создавая семафор, вы передаете ему начальное и максимальное значения счетчика, как показано ниже:

```
CSemaphore Semaphore(2, 2);
```

Посольку в этом разделе семафоры будут использоваться для создания потокового класса, логично будет объявить указатель на объект класса `CSemaphore` в качестве переменной-члена потокового класса, а затем динамически создать объект класса `CSemaphore` в конструкторе потокового класса, как показано ниже:

```
semaphore = new CSemaphore(2, 2);
```

Поступать так необходимо по той причине, что инициализацию переменной-члена класса потока необходимо выполнять в его конструкторе, а не при объявлении. При работе с объектами критической секции и защелки отсутствовала необходимость передачи конструктору их классов каких-либо аргументов. Поэтому можно было создавать объект их класса критической секции (или защелки) одновременно с объявлением класса потока.

Теперь, когда объект семафора создан, можно начинать отчет количества обращений к ресурсу. Для реализации процесса подсчета прежде всего необходимо создать экземпляр класса `CSingleLock` (или класса `CMultiLock`, если вы собираетесь работать с несколькими объектами синхронизации потоков), передав ему указатель на семафор, который вы хотите использовать:

```
CSingleLock singleLock(semaphore);
```

Затем для уменьшения значения счетчика семафора вызывается метод `Lock()` объекта класса `CSingleLock`:

```
singleLock.Lock();
```

На данный момент объект семафора уже выполнил уменьшение значения своего внутреннего счетчика. Это новое значение сохраняется до тех пор, пока объект семафора не будет освобожден посредством вызова его метода `Unlock()`:

```
singleLock.Unlock();
```

В альтернативном варианте, если вы локально создали объект класса `CSingleLock` в стеке, достаточно просто позволить этому объекту выйти из области видимости, в результате чего будет не только автоматически освобождена выделенная объекту память, но и выполнено освобождение захваченного семафора. Другими словами, и вызов `Unlock()`, и удаление из стека объекта класса `CSingleLock` приведут к увеличению значения счетчика семафора и тем самым дадут возможность ожидающему потоку получить доступ к защищенному ресурсу.

В листинге 27.11 приведен текст файла заголовка для класса CSomeResource. Класс CSomeResource не выполняет никаких действий и вызывается только в целях демонстрации использования семафоров. Класс имеет единственную переменную-член, являющуюся указателем иа объект класса CSemaphore. Кроме того, в классе определены конструктор и деструктор, а также метод UseResource(), в котором непосредственно используется семафор.

Листинг 27.11. Файл заголовка SomeResource.h

```
#include "afxmt.h"

class CSomeResource
{
private:
        CSemaphore* semaphore;

public:
        CSomeResource() {};
        ~CSomeResource() {};

 void UseResource();
};
```

В листинге 27.12 приведен текст файла, реализующего класс CSomeResource. Можно видеть, что объект класса CSemaphore динамически создается в конструкторе класса CSomeResource и уничтожается в его деструкторе. Метод UseResource() эмулирует доступ к ресурсу. Он захватывает семафор, затем ожидает пять секунд и вновь его освобождает. Последнее происходит, когда функция завершает свою работу и объект класса CSingleLock выходит из области видимости.

Листинг 27.12. Файл реализации класса SomeResource.cpp

```
#include "stdafx.h"
#include "SomeResource.h"

CSomeResource::CSomeResource()
{
        semaphore = new CSemaphore(2,2);
}

CSomeResource::~CSomeResource()
{
        delete semaphore;
}
Void CSomeResource::UseResource()
{
        CSingleLock singleLock(semaphore);
        SingleLock.Lock();

        Sleep(5000);
}
```

Если модифицировать приложение Thread так, чтобы в нем использовался объект класса CSomeResource, можно наблюдать работу семафоров на практике. Выполните следующие операции.

1. Удалите из проекта все файлы CountArray.
2. Создайте в проекте два новых пустых файла SomeResource.h и SomeResource.cpp.

3. Добавьте в эти пустые файлы тексты программ, приведенные в листингах 27.11 и 27.12 соответственно.

4. Откройте файл ThreadView.cpp и замените директиву

```
#include "CountArray2.h";
```

директивой

```
#include "SomeResource.h"
```

5. Замените строку

```
CCountArray2 countArray;
```

следующей строкой:

```
CSomeResource someResource;
```

6. Замените функции WriteThreadProc() и ReadThreadProc() функциями, текст которых приведен в листинге 27.13.

Листинг 27.13. Функции ThreadProc1(), ThreadProc2() и ThreadProc3()

```
UINT ThreadProc1(LPVOID param)
{
        someResource.UseResource();

        ::MessageBox((HWND)param,
                "Thread 1 had access.", "Thread 1", MB_OK);

        return 0;
}

UINT ThreadProc2(LPVOID param)
{
        someResource.UseResource();

        ::MessageBox((HWND)param,
                "Thread 2 had access.", "Thread 2", MB_OK);

        return 0;
}

UINT ThreadProc3(LPVOID param)
{
        someResource.UseResource();

        ::MessageBox((HWND)param,
                "Thread 3 had access.", "Thread 3", MB_OK);

        return 0;
}
```

7. Замените тело функции OnStartthread() текстом, приведенным в листинге 27.14.

Листинг 27.14. Новый текст функции OnStartthread()

```
HWND hWnd = GetSafeHwnd();
AfxBeginThread(ThreadProc1, hWnd);
AfxBeginThread(ThreadProc2, hWnd);
AfxBeginThread(ThreadProc3, hWnd);
```

Теперь откомпилируйте новую версию приложения Thread и запустите ее на выполнение. В раскрывшемся главном окне приложения выберите команду Thread⇨Start Thread. Приблизительно через пять секунд появятся два окна сообщений, информирующие о том, что первый и второй потоки получили доступ к защищенному ресурсу. Еще через пять секунд появится третье окно сообщений, в котором говорится о том, что третий поток также получил доступ к ресурсу. Третьему потоку потребовалось на пять секунд больше по той причине, что первые два потока первыми захватили контроль над ресурсом. Семафор в этой программе организован таким образом, что разрешает доступ к ресурсу только двум потокам одновременно. Таким образом, третий поток вынужден был ожидать, пока первый или второй поток освободит защищенный ресурс.

На заметку

Хотя учебные программы в этой главе демонстрируют использование единственного объекта синхронизвции потоков, в приложении может присутствовать любое число подобных объектов. Можно даже в одной и той же программе одновременно использовать критические секции, защелки и семафоры с целью обеспечения защиты различных наборов данных и ресурсов любыми способами.

ГЛАВА 28

Что еще полезно знать

В этой главе...

Создание консольных приложений

Создание и использование 32-битовых динамически связываемых библиотек

Сообщения и команды

Разработка программных продуктов, поддерживающих множество символьных наборов

Существует ряд тем, которые не были рассмотрены в каком-либо другом месте этой книги, но которые хорошо известны опытным программистам, использующим Visual C++. Их значительно проще воспринимать, если у вас есть опыт работы с Visual Studio и библиотекой классов MFC и техника программирования на C++ вам хорошо известна. В данной главе достаточно материала, чтобы показать, насколько интересны эти темы, и побудить вас изучить их самостоятельно.

Создание консольных приложений

Консольные приложения очень похожи на DOS-программы, хотя могут работать в окне произвольного размера. Они имеют строгий символьный интерфейс с курсором, управляемым клавишами клавиатуры, а не мышью. Для взаимодействия с пользователем в них используются системные вызовы Console API и символьные функции ввода-вывода, например `printf()` и `scanf()`.

Простое консольное приложение

Консольные приложения, как правило, запускаются из командной строки DOS или с помощью команды Start⇨Run с указанием полного имени требуемой программы. Пожалуй, консольные приложения можно отнести к числу самых простых в разработке программ, и эта версия компилятора непосредственно поддерживает их создание.

Давайте вместе выполним те несколько процедур, которые необходимы для создания типичного консольного приложения, а затем попытаемся найти преимущества, которые можно извлечь из создания подобных приложений. Первое консольное приложение, которое мы создадим, будет очередной версией классического `Hello, World!`, подаренного миру в 70-х годах Керниганом (Kernighan) и Ритчи (Ritchie) — создателями языка C, предшественника C++.

Чтобы создать консольное приложение, откройте Microsoft Visual Studio и выполните следующие операции.

1. В окне Visual Studio выберите команду File⇨New.
2. В диалоговом окне New щелкните на корешке вкладки Projects, чтобы раскрыть на экране уже знакомое нам диалоговое окно New Project (если вы с ним еще не знакомы, вернитесь к главе 1).
3. Назовите проект `HelloWorld`, укажите для его хранения подходящую папку и в расположенном слева списке выберите значение `Win32 Console Application`.
4. Щелкните на кнопке OK.

Проект будет создан немедленно — не будет никаких диалоговых окон мастера и никаких дополнительных вопросов. Теперь необходимо создать файлы исходного текста и заголовка, после чего включить их в проект. Весь наш пример будет размещаться в одном файле. Выполните следующие операции.

1. Выберите команду File⇨New и в раскрывшемся диалоговом окне щелкните на корешке вкладки File.
2. Оставьте флажок опции Add To Project установленным, чтобы новый файл был автоматически добавлен к проекту.
3. В списке слева выберите значение `C++ Source File`.
4. В поле имени файла введите `HelloWorld` (расширение `.cpp` будет добавлено автоматически).
5. В результате диалоговое окно New должно принять вид, показанный на рис. 28.1. Щелкните на кнопке OK.

Пустой текстовый файл с указанным именем будет создан и добавлен в проект всего за один шаг. Это можно считать значительным улучшением по сравнению с предыдущей версией Visual C++, где вам сначала нужно было создать файл с каким-либо именем, например Text1, затем переименовать его в HelloWorld.cpp (или присвоить ему другое требуемое имя) и, наконец, добавить его в проект.

Введите в новый файл текст программы, приведенный в листинге 28.1.

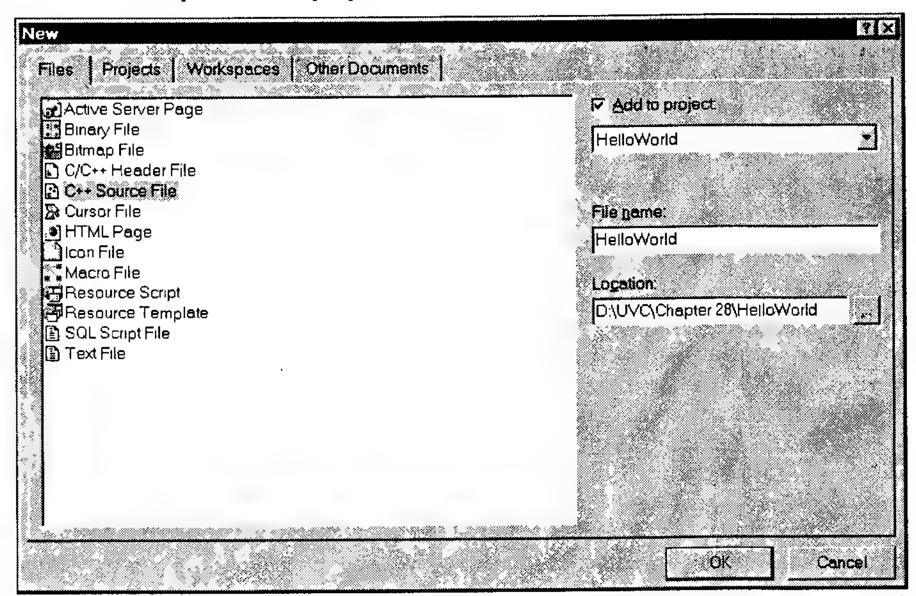


Рис. 28.1. Создание файла исходного текста для консольного приложения на C++

Листинг 28.1. Файл HelloWorld.Cpp

```
#include <iostream.h>
int main()
{
        cout << "Hello from the console!" << endl;
        return 0;
}
```

Выберите команду Build⇨Execute, чтобы скомпилировать, скомпоновать и запустить программу. На экране должно раскрыться окно DOS, показанное на рис. 28.2. Строка Press any key to continue... (Нажмите любую клавишу для продолжения...) генерируется системой, чтобы дать вам возможность увидеть результат работы программы до того, как окно DOS будет закрыто.

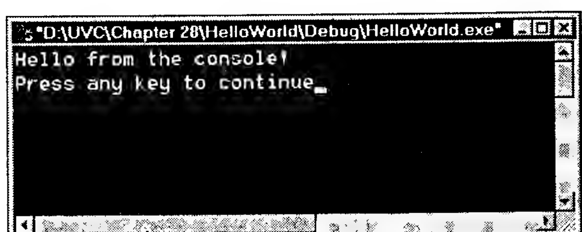


Рис. 28.2. Созданное приложение имеет интерфейс программы DOS

Объектно-ориентированное консольное приложение

Приложение Hello World написано на чистом языке C++ и не будет компилироваться компилятором С, который не поддерживает потоковый ввод-вывод с помощью оператора cout. Однако это приложение не является объектно-ориентированным, поскольку в нем отсутствуют объекты. Замените текст в файле HelloWorld.Cpp текстом, представленным в листинге 28.2.

Листинг 28.2. Файл HelloWorld.Cpp — использование объектов

```
// HelloWorld.cpp
//

#include <iostream.h>
#include <afx.h>

class Hello
{
private:
	CString message;

public:
	Hello();
	void display();
};

Hello::Hello()
{
	message = "Hello from the console!";
}

void Hello::display()
{
	cout << message << endl;
}

int main()
{
	Hello hello;
	hello.display();

	return 0;

}
```

Теперь у нас есть настоящая объектно-ориентированная программа. Более того, она использует класс CString из библиотеки MFC. С этой целью к ней подключается файл заголовка afx.h. Если вы сейчас сделаете попытку скомпилировать проект, то компоновщик выдаст сообщение об ошибке, относящееся к переменным _beginthreadex и _endthreadex. По умолчанию консольные приложения являются однозадачными, а библиотека MFC рассчитана на создание многозадачных приложений. Подключение файла afx.h и переход к работе с MFC превратили наше приложение в многозадачное, что несовместимо с установкой в проекте, принятой по умолчанию. Чтобы исправить это недоразумение, выберите команду Project⇨Settings и в раскрывшемся окне щелкните на корешке вкладки C/C++. В раскрывающемся списке, расположенном в верхней части диалогового окна, выберите значение Code Generation. В раскрывающемся списке User Runtime Library выберите значение Debug Multithreded. (Окончательный вид этого диалогового окна показан на рис. 28.3.) Щелкните на кнопке OK и заново скомпилируйте проект.

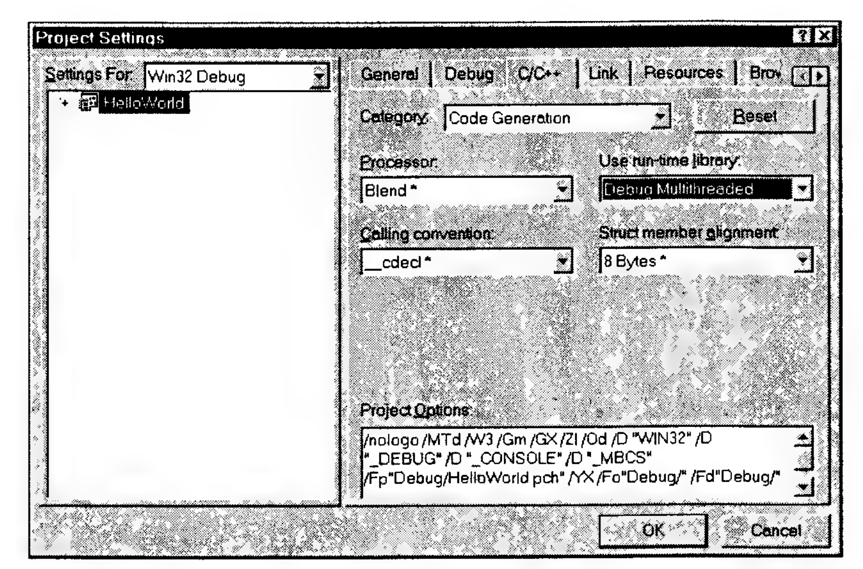


Рис. 28.3. При использовании библиотеки классов MFC необходимо обеспечить в приложении поддержку многозадачности

Результаты работы этой объектно-ориентированной программы не отличаются от результатов работы предыдущей версии программы — ведь это просто пример. Но теперь вы убедились, что в консольных приложениях можно использовать библиотеку MFC, что такие приложения можно создавать с использованием объектов и при этом приложение сохраняет достаточно небольшой объем. Они обязаны иметь функцию `main()`, так как именно она вызывается операционной системой, когда вы запускаете приложение.

На заметку Хотя это приложение само по себе невелико, Visual C++ создает много вспомогательных файлов. Поэтому каталог `Debug` имеет размер около 7,8 Мбайт, из которых почти 1,3 Мбайт занимает файл `HelloWorld.exe`. Остальные файлы являются библиотеками MFC, а они тоже не маленькие.

Автономное тестирование фрагментов программ

Лучшим аргументом в пользу создания DOS-приложений в настоящее время является потребность в автономном тестировании (scaffolding) небольших фрагментов программы или отдельных объектов. В подобных случаях возникает необходимость создания для тестируемого программного фрагмента временной рабочей оболочки. Наипростейшей рабочей оболочкой является консольное приложение, подобное тому, которое мы только что создали.

Для автономного тестирования объекта или функции необходимо выполнить следующее.

1. Создать новое консольное приложение, специально предназначенное для автономного тестирования.
2. Добавить функцию `main()` в файл с расширением `.cpp`, содержащий программный фрагмент, который необходимо тестировать.
3. Подключить файл заголовка для тестируемого объекта или функции.
4. Написать текст программы, тестирующей работу функции или функционирование объекта в различных режимах, и включить его в функцию `main()`.

Выполнив все это, вы получите возможность основательно протестировать программу, сосредоточив все внимание только на корректной работе и характеристиках производительности небольших фрагментов большого проекта. Автономное тестирование отвечает одному из канонов разработки программных продуктов, утверждающему, что *проектирование сосредоточивается на общем, а программирование — на частном.*

Применение автономного тестирования к любому алгоритму помогает убедиться в точности реализации деталей. Не забывайте о существовании и дополнительных преимуществ — включение тестируемого фрагмента в специально разрабатываемый модуль дает вам возможность четко задокументировать, как именно тестировался этот фрагмент и как правильно его использовать. Все это может очень пригодиться в процессе дальнейшего тестирования, отладки или расширения приложения.

Создание и использование 32-битовых динамически связываемых библиотек

Динамически связываемые библиотеки (DLL-модули) являются базовым компонентом операционных систем Windows 95 и Windows NT. Windows 95 использует модули `Kernel32.dll`, `User32.dll` и `Gdi32.dll` для выполнения абсолютного большинства своих функций. И вы тоже можете ими пользоваться. Электронная документация Microsoft Visual C++ может служить хорошим источником информации о функциях API, реализованных в этих трех модулях DLL.

Еще одним инструментом для получения информации о приложениях Windows является утилита `DumpBin`, находящаяся в папке `\Program Files\Microsoft Visual Studio\VC98\BIN`. Эта утилита представляет собой программу с управлением из командной строки, которая выводит информацию об импорте и экспорте выполняемых файлов и динамически связываемых библиотек.

Модули DLL Windows можно использовать в любой разрабатываемой программе. Кроме того, интегрированная среда Visual Studio позволяет создавать и свои собственные модули DLL.

Создание 32-битового модуля DLL

В Visual C++ существует два вида модулей DLL — использующие библиотеку MFC и не использующие ее. Как вы скоро узнаете, для каждого из видов модулей DLL имеется свой собственный мастер.

Если собрать несколько функций в один модуль DLL, то данный модуль будет *экспортировать* эти функции для использования в других программах. И довольно часто одни модули DLL для выполнения своей работы нуждаются в *импорте* функции из других модулей DLL.

Импорт и экспорт функций

Для определения экспортируемой функции используется следующий синтаксис:

```
__declspec(dllexport) типДанных int идентификаторПеременной; // Для переменных.
```

или

```
__declspec(dllexport) возвращаемыйТип имяФункции ( [списокАргументов] ); // Для
функций.
```

Импортирование функций организуется практически так же — просто замените ключевое слово, например __declspec(dllexport), словом __declspec(dllimport). Используя реальные функцию и переменную для демонстрации синтаксиса, получим следующее:

```
__declspec(dllimport) int referenceCount;
__declspec(dllimport) void DiskFree( lpStr Drivepath );
```




Ключевому слову __declspec предшествуют два знака подчеркивания.

Чтобы упростить описание DLL-модулей, Microsoft использует файл заголовка и макросы препроцессора. Эта методика всего лишь требует, чтобы вы использовали уникальную лексему препроцессора — проще всего для этого использовать имя файла заголовка — и написали макрос, который будет замещать эту лексему корректными операторами импорта и экспорта. Предположим, что имеется файл заголовка с именем DISKFREE.H, тогда макрос препроцессора в нем может выглядеть так, как показано в листинге 28.3.

Листинг 28.3. Файл заголовка DiskFree.h

```
// DISKFREE.H содержит простую функцию, возвращающую размер
// свободного дискового пространства.
#ifndef __DISKFREE_H
#define __DISKFREE_H

#ifndef __DISKFREE__
#define DISKFREELIB __declspec(dllimport)
#else
#define DISKFREELIB __declspec(dllexport)
#endif
//Макрос используется для выбора описания импорта или экспорта.
DISKFREELIB unsigned long DiskFree( unsigned int drive );
//Например, 0 = A:, 1= B:, 2 = C:
#endif
```

Подключив файл заголовка, вы даете возможность препроцессору определить, импортируется или экспортируется функция DiskFree. Теперь вы можете предоставить такой файл заголовка как разработчику, так и пользователю модуля DLL, что решает много проблем при сопровождении программ.

Создание модуля DLL DiskFree

Утилита DiskFree позволяет легко определять размер свободного пространства на указанном диске. Ее работа основана на использовании функции GetDiskFreeSpace(), находящейся в модуле Kernel32.Dll.

Для того чтобы создать не использующий MFC модуль DLL, выберите команду File⇨New, щелкните на корешке вкладки Project и выберите в расположенном слева списке значение Win32Dll. Назовите проект DiskFree и щелкните на кнопке OK. На экране появится окно мастера AppWizard, как показано на рис. 28.4. Выберите переключатель An empty DLL project, и в результате будет создан проект, пока еще не имеющий собственных файлов.

Добавьте в проект файл заголовка DiskFree.h и поместите в него текст, представленный в листинге 28.4. Добавьте в проект файл текста программы DiskFree.cpp и поместите в него текст, представленный в листинге 28.5.

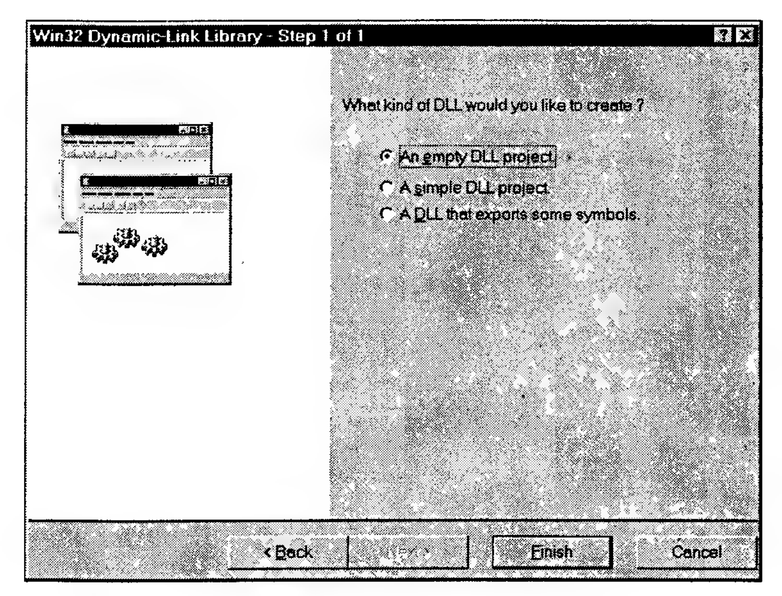


Рис. 28.4. Создание проекта модуля DLL, не использующего MFC, выполняется за один шаг

Листинг 28.4. Файл заголовка `DiskFree.h`

```
#ifndef __DISKFREE_H
#define __DISKFREE_H
#ifndef __DISKFREE__
#define __DISKFREELIB__ __declspec(dllimport)
#else
#define __DISKFREELIB__ __declspec(dllexport)
#endif
// Возвращает размер свободного дискового пространства на диске,
// заданном его номером (например, 0 = A:, 1= B:, 2 = C:).
__DISKFREELIB__ unsigned long DiskFree( unsigned int drive );
#endif
```

Листинг 28.5. Файл исходного текста `DiskFree.cpp`

```
#include <afx.h>
#include <winbase.h>     // Содержит объявление функции GetDiskFreeSpace(),
                                   // находящейся в модуле kernel32.dll.
#define __DISKFREE__ // Определяет лексему перед подключением библиотеки.
#include "diskfree.h"
// Возвращает размер свободного дискового пространства на диске,
// заданном его номером (например, 0 = A:, 1= B:, 2 = C:).
__DISKFREELIB__ unsigned long DiskFree( unsigned int drive )
{
        unsigned long bytesPerSector, sectorsPerCluster,
                freeClusters, totalClusters;
        char DrivePath[4] = { char( drive + 65 ), :, \\, \0 };
        if( GetDiskFreeSpace( DrivePath, &sectorsPerCluster,
                &bytesPerSector, &freeClusters, &totalClusters ))
        {
                return sectorsPerCluster * bytesPerSector * freeClusters;
```

```
        }
        else
        {
                return 0;
        }
}
```

Теперь можно оттранслировать модуль DLL. В следующем разделе речь будет идти об использовании 32-битовых модулей DLL и о том, как Windows находит модули DLL в системе.

Наиболее общим назначением модулей DLL является обеспечение повторного использования и расширение функциональной поддержки программного обеспечения при возможности их неявной загрузки в среде Windows. Ниже перечислены темы, которые не были освещены в этой книге, но которые вы, возможно, захотите изучить самостоятельно.

- Выбор между динамической и статической компоновками классов MFC
- Выбор между неявной и явной загрузками модулей DLL, причем последняя требует использования функций `LoadLibrary()` и `FreeLibrary()`
- Многозадачные модули DLL
- Обеспечение совместного доступа к данным извне модулей DLL
- Соглашения по вызовам функций модулей DLL из других языков программирования (`__stdcall`, `WINAPI` и т.д.)

В этой главе при компиляции модуля `DiskFree` использовались установки по умолчанию. Это означает, что модуль DLL `Main` будет применяться неявно (его добавляет компилятор) и что используется режим неявной загрузки модулей DLL, причем Windows будет автоматически управлять загрузкой и выгрузкой этих библиотек.

Использование 32-битовых модулей DLL

Большинство модулей DLL загружается неявно, и их загрузкой и выгрузкой управляет Windows. Процедура поиска библиотек, загружаемых подобным образом, аналогична процедуре поиска выполняемых файлов — сначала поиск происходит в папке, из которой было загружено использующее их приложение, затем выполняется поиск в текущей папке, затем — в папке `Windows\System`, затем — в папке `Windows` и наконец — в каждой из папок, указанных в переменной `PATH`.

Обычно при установке приложения модули DLL принято помещать в папку `Windows` или `Windows\System`. Но в процессе разработки приложения в качестве временного хранилища для этой цели можно использовать папку выполняемых файлов проекта. Однако нужно следить за тем, чтобы в итоге в разных папках, включая папки `Windows`, `Windows\System` и папку проекта, не оказались разные версии одного и того же модуля DLL.

Использование модуля DLL

Неявная загрузка и использование модуля DLL осуществляются так же просто, как и работа с обычными функциями. Это особенно заметно, если вы построите файл заголовка так, как рассказывалось в предыдущем разделе. При компиляции модуля DLL транслятор Microsoft Visual C++ создает файл с расширением `.LIB`. (Таким образом, помимо файла `DISKFREE.DLL`, существует еще и файл `DISKFREE.LIB`, созданный компилятором.) Файл библиотеки (`.LIB`) используется для разрешения адресов загрузки в модуле DLL и содержит полное имя динамически подключаемой библиотеки, тогда как файл заголовка содержит его описание.

Все, что вам необходимо сделать, — это подключить файл заголовка к исходному файлу, использующему функции из модуля DLL, и указать имя файла библиотеки (.LIB) в поле Object/library modules на вкладке Link диалогового окна Project Settings, доступ к которому можно получить, выбрав команду Project⇨Settings (рис. 28.5).

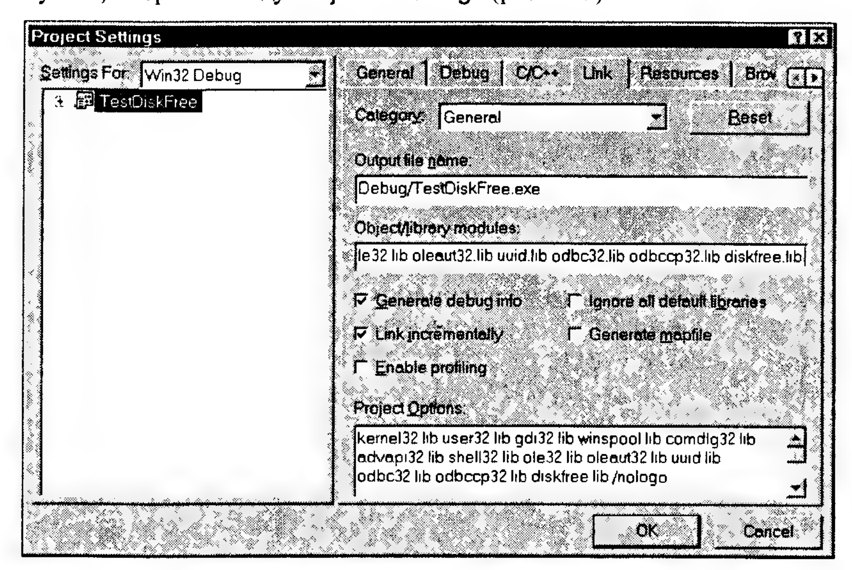


Рис. 28.5. Добавление в установки проекта файла библиотеки (.LIB)

Для проверки работы модуля DLL DiskFree создайте консольное приложение с именем TestDiskFree и добавьте в него файл TestDiskFree.cpp исходного текста на С++. Внесите в файл текст программы, представленный в листинге 28.6. Скопируйте в папку этого проекта файл DiskFree.h и добавьте его к проекту, выбрав команду Project⇨Add To Project⇨Files, а затем выбрав файл DiskFree.h. Скопируйте в папку TestDiskFree файлы DiskFree.Dll и DiskFree.Lib. (Они находятся в папке DiskFree\Debug.) Выполните все указанные выше изменения в установках проекта и скомпилируйте его.

Листинг 28.6. Файл DiskFree.Cpp

```
#include <afx.h>
#include <iostream.h>
#include "diskfree.h"
#define CodeTrace(arg) \
        cout << #arg << endl;\
        arg
int main()
{
        CodeTrace( cout << DiskFree(2) << endl );
        return 0;
}
```

Эта программа неявно загружает модуль DLL посредством подключения файла diskfree.h, а затем использует его в работе. При выполнении программы макрос CodeTrace просто печатает оператор программы перед его выполнением. Фактически вся работа приложения состоит в вызове функции DiskFree() с целью определения имеющегося свободного дискового пространства на диске 2. Диск 0 — это A:, диск 1 — B: и диск 2 — C:. Результаты работы скомпилированной и запущенной программы должны быть подобны тому, что представлено на рис. 28.6.

Согласно программе TestDiskFree на диске C: компьютера, использованного для запуска этой программы, находится почти 37 Мбайт свободного дискового пространства. Это соответствует действительности. Теперь вы знаете, как разработать и поместить в модуль DLL реальные функции, а также, как их использовать или же сделать доступными для других пользователей.

Рис. 28.6. Созданная программа осуществляет вызов функции из модуля DLL

Сообщения и команды

Как уже отмечалось в главе 3, сообщения являются ключевым компонентом операционной системы Windows. Все, что происходит в приложениях Windows, происходит в результате обработки соответствующих сообщений. Когда вы перемещаете указатель мыши и щелкаете на кнопке, отображенной на экране, генерируется огромное количество сообщений, включая WM_MOUSEMOVE — при каждом перемещении мыши, WM_LBUTTONDOWN — при каждом нажатии ее кнопки, WM_LBUTTONCLICK — при каждом отпускании ее кнопки. На верхнем уровне операционной системы генерируются более абстрактные сообщения, например сообщение WM_COMMAND, имеющее в качестве одного из параметров идентификатор ресурса кнопки, на которой был сделан щелчок. При желании низкоуровневые сообщения можно игнорировать, что и делает большинство программистов.

Возможно, вам еще не известно, что генерировать сообщения может и ваша программа. Существует две функции, предназначенные для генерации сообщений, — CWnd::SendMessage() и CWnd::PostMessage(). Каждая из них передает сообщение объекту, являющемуся потомком класса CWnd. Объект, который собирается послать сообщение в некоторое окно с помощью одной из данных функций, должен иметь указатель на это окно, а окно должно быть подготовлено к приему такого сообщения. Самым распространенным приемом в такой ситуации будет использование в посылающем сообщение объекте переменной-члена для хранения указателя на окно, которому сообщение будет адресовано, и еще одной переменной-члена, предназначенной для хранения посылаемого сообщения:

```
CWnd* m_messagewindow;
UNIT m_message;
```

Сообщения представляются целыми числами без знака. Они имеют имена только потому, что с помощью директив #define целые числа поставлены в соответствие именам типа WM_MOUSEMOVE.

Класс — отправитель сообщения имеет специальный метод для инициализации этих переменных и он, как правило, невелик по размеру:

```
void Sender::SetReceiveTarget(CWnd *window, UINT message)
{
        m_messagewindow = window;
        m_message = message;
}
```

Когда классу-отправителю необходимо отослать сообщение в некоторое окно, он вызывает метод SendMessage():

```
m_messagewindow->SendMessage (m_message, wparam, lparam);
```

Существует и другая возможность — вызвать функцию PostMessage():

```
m_messagewindow->PostMessage (m_message, wparam, lparam);
```

Разница между ними заключается в том, что функция SendMessage() не возвращает управление до тех пор, пока сообщение не будет обработано окном, которому оно адресовано, а функция PostMessage() просто помещает его в очередь сообщений и сразу же завершает свою работу. Если, к примеру, вы создаете объект, передаете указатель на него в качестве аргумента lparam в сообщении, а затем намереваетесь удалить такой объект, то в этом случае следует использовать функцию SendMessage(), так как вам нельзя удалять объект до тех пор, пока механизм обработки сообщений не обработает его. Если же вы не передавали указатель, то, вероятно, можно использовать PostMessage() и продолжить работу, как только сообщение будет добавлено в очередь.

Смысл значений wparam и lparam зависит от посылаемого сообщения. Если этим определяемым системой сообщением является, скажем, WM_MOUSEMOVE, то информацию о его параметрах вы сможете найти в электронной документации. Если, что более вероятно, вы посылаете свое собственное сообщение, то смысл этих параметров зависит полностью от вас. Именно вы являетесь автором сообщения и соответственно подпрограммы его обработки в классе окна-адресата.

Для создания сообщения необходимо добавить в файл заголовка класса-адресата следующую директиву:

```
#define WM_HELLO WM_USER+300
```

WM_USER — это целое число без знака, которое указывает на начало диапазона номеров сообщений, выделенных для сообщений, определяемых пользователем. В этой версии библиотеки MFC его значение равно 0x4000, но вам не следует непосредственно использовать такое значение. Определяемые пользователем сообщения должны иметь номера от WM_USER до 0x7FFF.

Затем необходимо добавить строку в карту сообщений как в файле заголовка, так и в файле исходного текста, поместив ее вне комментария, сформированного ClassWizard. Карта сообщений в файле исходного текста может выглядеть следующим образом:

```
BEGIN_MESSAGE_MAP(CMainFrame, CMDIFrameWnd)
        //{{AFX_MSG_MAP(CMainFrame)
                // ВНИМАНИЕ: ClassWizard будет размещать здесь макросы карты сообщений.
                // НЕ РЕДАКТИРУЙТЕ текст, автоматически помещенный в этот блок!
        //}}AFX_MSG_MAP
        ON_MESSAGE(WM_HELLO, OnHello)
END_MESSAGE_MAP()
```

Строка, добавленная после комментария //AFX_MSG_MAP, перехватывает сообщение WM_HELLO и организует вызов функции OnHello(). Карта сообщений файла заголовка может выглядеть следующим образом:

```
// Сгенерированные функции карты сообщений
protected:
        //{{AFX_MSG_MAP(CMainFrame)
        afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
                // ВНИМАНИЕ: ClassWizard будет размещать здесь функции-члены.
                // НЕ РЕДАКТИРУЙТЕ текст, автоматически помещенный в этот блок!
        //}}AFX_MSG_MAP
        afx_msg LRESULT OnHello(WPARAM wParam, LPARAM lParam);
        DECLARE_MESSAGE_MAP()
```

Для завершения процесса остается лишь добавить в исходный файл реализацию функции OnHello().

Разработка программных продуктов, поддерживающих множество символьных наборов

Границы между государствами имеют все меньшее значение. Доступ к средствам коммуникации постоянно расширяется, соответственно все больше возрастает спрос на программные продукты, созданные производителями по всему миру. Даже при разработке программного обеспечения "для домашнего употребления" уже нельзя не принимать во внимание аспект интернационализации такого рода продукции. Рост популярности Internet способствовал расширению рынка программного обеспечения на страны, в которых английский не является родным языком и набор символов занимает вторую часть кодовой страницы стандарта ASCII. Все сказанное выше означает, что создаваемое вами приложение должно быть способно к общению с пользователем на языке, отличном от английского, при использовании набора символов, отличающегося от принятого на Западе.

Микрокомпьютеры были созданы в Соединенных Штатах, что объясняет, почему операционные системы базируются на 8-битовых наборах символов. В английском алфавите 26 букв, а цифр всего десять, и поэтому остается достаточно много места (до 220 символов) для знаков пунктуации и прочих символов. Но в некоторых странах, таких как Япония и Китай, необходимы наборы из тысяч символов.

Unicode — это один из способов решения проблем, связанных с символьными наборами. Стандарт Unicode был разработан и поддерживается консорциумом, в который входят крупнейшие субъекты международного компьютерного рынка. Среди них — фирмы Adobe, Aldus, Apple, Borland, IBM, Lotus, Microsoft, Novell и Xerox.

В стандарте Unicode для кодирования каждого символа используются два байта, тогда как в ASCII — только один. Одним байтом (8 бит) может кодироваться 2^8 или 256 символов. Двумя байтами (16 бит) можно закодировать 65 536 символов. Этого достаточно не только для одного языка, но и для всех широко используемых наборов символов вместе взятых. К примеру, для кодирования японской азбуки, одной из самых больших, требуется приблизительно 5 000 символов. Для большинства остальных языков их требуется значительно меньше. В спецификации Unicode для различных символьных наборов установлены различные диапазоны значений, поэтому она охватывает почти все языки одним универсальным кодом — Unicode.

Библиотека MFC полностью поддерживает Unicode, включая версии Unicode почти для всех функций. Например, рассмотрим функцию `CWnd::SetWindowText()`. Ей передается строковая переменная, значение которой эта функция присваивает либо заголовку окна, либо названию кнопки. Тип передаваемой строковой переменной зависит от того, включена ли в приложение поддержка Unicode. На самом деле существует две различные версии функций для работы с текстом в окне — версия с поддержкой и версия без поддержки Unicode. В файле `WINUSER.H` фрагмент текста, приведенный в листинге 28.7, изменяет имя вызываемой функции на `SetWindowTextW`, если установлена поддержка Unicode, или на `SetWindowTextA` в противном случае.

Листинг 28.7. Фрагмент файла WINUSER.H, в котором реализована поддержка набора символов Unicode

```
WINUSERAPI BOOL WINAPI SetWindowTextA(HWND hWnd, LPCSTR lpString);
WINUSERAPI BOOL WINAPI SetWindowTextW(HWND hWnd, LPCWSTR lpString);

#ifdef UNICODE
#define SetWindowText SetWindowTextW
#else
#define SetWindowText SetWindowTextA
#endif //!UNICODE
```

Отличие между этими двумя функциями заключается в типе второго параметра: LPCSTR для A-версии и LPCWSTR — для W-версии (*Wide — расширенный*).

Если вы используете Unicode, то везде, где вы передаете функции строковый литерал (подобно "Hello"), поместите его в макросе _T, как показано ниже:

```
pWnd->SetWindowText(_T("Hello"));
```

Если усилием воли вы заставите себя выполнить весьма утомительную процедуру размещения всех строковых констант в макросе _T, то будете вознаграждены: созданное приложение сможет работать с Unicode. Теперь для вас не составит большого труда подготовить версию приложения на японском языке или на суахили, если на то будет желание заказчика.

На заметку Windows 95 основана на прежней Windows, и по этой причине в ней не реализована поддержка Unicode. Это означает, что при работе с Unicode в приложениях Windows 95 вы будете ощущать снижение производительности из-за того, что ядро Windows 95 будет вынуждено конвертировать строки Unicode в обычные строки. Операционная система Windows NT разрабатывалась фирмой Microsoft совершенно независимо и полностью совместима со стандартом Unicode.

Если вы на C++ разрабатываете программы с использованием Unicode для нескольких платформ, версии для Windows 95 могут показаться вам работающими очень медленно по сравнению с версиями для Windows NT.

Пустая страница



Приложения

В этой части...



Обзор языка C++ и основные концепции объектно-ориентированного программирования

В этом приложении...

- Работа с объектами
- Повторное использование кода и наследование
- Управление памятью

Работа с объектами

Язык C++ представляет собой язык объектно-ориентированного программирования. Хотя с его помощью можно писать и простейшие программы типа Hello World, о которой шла речь в главе 28, по-настоящему он проявляет себя при создании больших приложений. Главная отличительная черта языков объектно-ориентированного программирования заключается в том, что они трактуют приложение как *объект*, а не как *процедуру*. Программирование на Visual C++ предполагает широкое использование библиотеки MFC, которая предоставляет средства для реализации практически любых видов объектов, в которых может возникнуть необходимость при разработке приложения.

Что такое объект

Объект — это совокупность отдельных информационных элементов и функций, которые ими оперируют. Например, объект BankAccount (БанковскийСчет) должен соединить воедино номер клиента, номер счета и текущий остаток на счете — это три информационных элемента, которые в совокупности необходимы для осуществления операций с банковским счетом. Средства группирования связанной информации присутствуют во многих языках в виде *структур* (structure) или *записей* (record). Однако коренное отличие объектов от таких средств группирования состоит в том, что, помимо информационных элементов, объект содержит и *функции*, определяющие его поведение в информационной среде. Например, наш объект BankAccount может содержать функции Deposit() (ПоложитьНаСчет), Withdraw() (СнятьСоСчета) и GetBalance() (ПодсчитатьБаланс). На рис. А.1 схематически показана структура объекта.

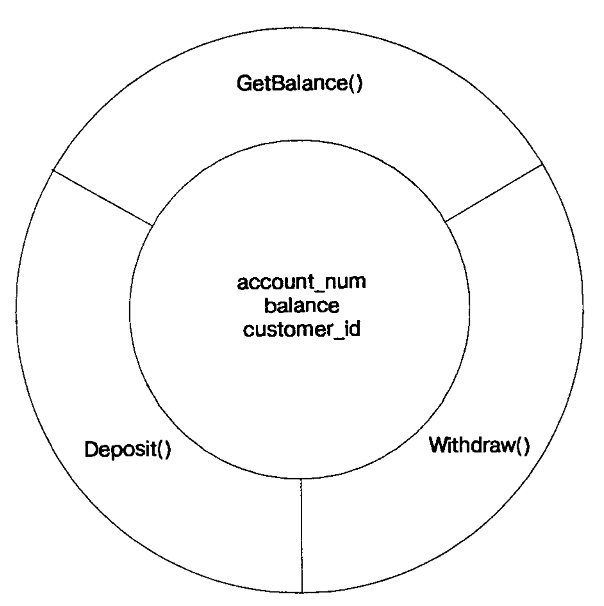


Рис. А.1. Объект объединяет информацию (переменные) и поведение (функции)

Для чего нужны объекты

У объектно-ориентированного подхода есть много привлекательных черт, но главными, пожалуй, являются удобство сопровождения и жесткость программной конструкции. Задумайтесь о том, как вы будете убеждать руководство перейти на язык C++ при разработке программ. Скорее всего, главный довод таков: при изменении какой-либо одной функциональной возможности в программе не нужно пересматривать все компоненты; не нужно задумываться над тем, как отреагирует операция B на изменения в операции A. Оба упомянутых преимущества вытекают из того, что любое обращение извне к информации, хранимой внутри объекта, возможно только посредством включенных в состав объекта функций. Например, в некотором фрагменте программы создается объект банковского счета:

```
BankAccount accont;
```

Операции зачисления на счет некоторой суммы, снятия со счета и подсчета остатка будут выглядеть в программе следующим образом:

```
accont.Deposit(100.00);
accont.WithDraw(50.00);
float newbalance = accont.GetBalance();
```

Нельзя реализовать те же операции следующим образом:

```
accont.balance = 100.00;
accont.balance -= 50.00;
float newbalance = accont.balance;
```

Метод сокрытия информации предназначен для того, чтобы программист сосредоточивал свое внимание на главном — операциях с объектом, а не второстепенных деталях (способах их реализации внутри объекта).

Предположим, например, что сначала вы решили хранить текущий остаток на счете в формате с плавающей точкой (в качестве единицы был взят доллар), а со временем пришли к выводу, что операции с целыми числами (например, в центах) будут выполняться быстрее и для хранения требуется меньше места в памяти, поэтому имеет смысл перейти на такой формат хранения данных. Для изменения программы придется модифицировать функции `Deposit()`, `Withdraw()` и `GetBalance()`, которые должны преобразовывать аргументы из формата `float` в формат `int` и пересчитывать доллары в центы (и наоборот). После этого все модификации можно считать завершенными. Те ваши коллеги, которые занимались другими частями проекта, даже не узнают о внесенных изменениях, поскольку для них способ общения с объектом остался прежним. А вот если бы вы использовали в программе второй метод доступа к компонентам объекта, пришлось бы искать по всей программе фрагменты с обращением к переменной `balance` и корректировать соответствующие операторы.

Ну а если вы так хорошо продумали структуру данных, что формат данных никогда изменять не придется? Имеет смысл ли в этом случае использовать средства объектно-ориентированного программирования? Ответ также утвердительный. На практике очень часто приходится сталкиваться со случаем, когда в процессе эксплуатации программы меняются те или иные правила выполнения операций, запрограммированных в ней. Например, в нашем случае со временем руководство пришло к выводу, что нужно изменить правило снятия суммы со счета, блокируя перерасход:

```
balance -= ammounttowithdraw;
if( balance < 0)
      balance += ammounttowithdraw; // Отменить снятие со счета.
```

А затем у руководства возникла новая идея — разрешить клиентам некоторый кредит (overdraftlimit). Тогда алгоритм снятия со счета будет выглядеть так:

```
balance -= ammounttowithdraw;
if( balance < overdraftlimit)
      balance += ammounttowithdraw; // Отменить снятие со счета.
```

Если все эти изменения будут происходить только внутри функции Withdraw(), то они пройдут безболезненно для всего приложения и новая дисциплина обслуживания клиентов будет внедрена в считанные минуты. Если же в программе использовалось прямое обращение к компонентам объекта, придется потрудиться, вылавливая все операции снятия со счета в программе и корректируя их.

Понятие класса

В любом банке счетов столько, сколько вкладчиков — тысячи. Все они имеют аналогичную структуру и аналогичные операции над хранящимися в них данными. О таких счетах в программе обслуживания банка говорят как об *объектах (экземплярах)* класса BankAccount.

При определении класса нужно описать его сущность — состав данных, характеризующих объект этого класса, и операции, которые могут быть с этими данными выполнены. Первые представляют собой *члены-переменные* класса, а вторые — *члены-функции* (или *методы*). Нужно также определить, какие из членов желательно защитить от внешнего воздействия для реализации принципа сокрытия информации. В листинге А.1 показано определение класса BankAccount.

Листинг А.1. Определение класса BankAccount

```
class BankAccount
{
      private:
            float balance;
            char[8] customer_id;
            char[8] account_num;
      public:
            float GetBalance();
            void WithDraw(float ammounttowithdraw);
            void Deposit(float ammounttodeposit);
};
```

Ключевое слово private перед идентификаторами членов-переменных предназначено для компилятора. Последний откажется компилировать код, в котором есть прямое обращение к этой переменной, если только данный код не принадлежит методу того же класса BankAccount. Ключевое слово public также предназначено для компилятора и позволяет последнему компилировать любое обращение к перечисленным за ним членам класса (в данном случае — методам). Мы привели в этом листинге классический пример организации методов доступа к членам класса: все переменные являются закрытыми (private), а все методы — открытыми (public).



Иногда методу поручаются некоторые повторяющиеся вспомогательные операции над членами класса, причем нежелательно, чтобы такие операции инициировались извне. В этом случае метод также объявляется как закрытый (private).

Теперь после объявления класса допустим, что в программе уже объявлены два объекта этого класса — mine и yours, каждый из которых имеет собственные члены balance, customer_id и account_num. Если положить деньги на счет объекта mine, это никак не отразится на величине текущего остатка (значении члена balance) счета yours. В листинге А.2 показан фрагмент программы, в котором создаются объекты класса BankAccount и вызываются их методы.

Листинг А.2. Использование объектов класса BankAccount

```
BankAccount mine, yours;
mine.Deposit(1000);  ,
yours. Deposit(100);
mine.WithDraw(500)
float mybalance = mine.GetBalance();
float yourbalance = yours.GetBalance();
```

Реализация методов

Все три метода класса BankAccount — Deposit(), Withdraw() и GetBalance() — должны быть реализованы, т.е. подготовлен их программный код; затем он должен быть откомпилирован. Код реализации можно разместить либо внутри блока объявления класса (такой код называется *встроенным* — *inline*), либо вне объявления класса (как правило — в отдельном файле реализации). Встроенными объявляются только очень простые методы, как в нашем классе BankAccount (листинг А.3).

Листинг А.3. Определение встроенных методов класса BankAccount

```
class BankAccount
{
	private:
		float balance;
		char[8] customer_id;
		char[8] account_num;
	public:
		float GetBalance(){return balance;}

		void WithDraw(float ammounttowithdraw)
		{
			balance -= ammounttowithdraw;
			if( balance < 0)
				balance += ammounttowithdraw; // Отменить снятие со 
счета.
		}

		void Deposit(float ammounttodeposit){balance += ammounttodeposit;}
};
```

Обратите внимание на то, что точка с запятой после имени функции в листинге А.1 заменена телом функции, заключенным в фигурные скобки. Функция Withdraw() несколько длиннее остальных, а потому ее лучше было бы реализовать в отдельном файле. В этом случае нужно перед именем функции обязательно указать имя класса, членом которого она является. В листинге А.4 показан программный код реализации Withdraw() в случае, если он будет размещен вне определения класса.

Листинг А.4. Реализация Withdraw() вне определения класса

```
void BankAccount::WithDraw(float ammounttowithdraw)
        {
                balance -= ammounttowithdraw;
                if( balance < 0)
                        balance += ammounttowithdraw; // Отменить снятие со счета.
        }
```

На заметку Как правило, определение класса размещается в отдельном файле заголовка, имя которого соответствует имени класса. Например для нашего класса BankAccount должен быть создан файл BankAccount.h. Остальной же код размещается в другом файле — *файле реализации* с расширением .cpp. В нашем случае файл реализации этого класса будет назван BankAccount.cpp.

Встроенные функции

Нужно разделять встроенный код, как в листинге А.3, и *встроенные функции*. Хороший компилятор часто самостоятельно формирует встроенные функции, если выясняется, что ее код достаточно компактен. Это приводит к некоторому повышению скорости выполнения программы. Если же формируется встроенный код, то компилятор поневоле транслирует эту функцию как встроенную. Дать указание компилятору рассматривать некоторую функцию, реализованную вне определения класса, как встроенную, можно при помощи ключевого слова inline, например:

```
inline void BankAccount::WithDraw(float ammounttowithdraw)
        {
                balance -= ammounttowithdraw;
                if( balance < 0)
                        balance += ammounttowithdraw; // Отменить снятие со счета.
        }
```

Если предполагается, что данную функцию будут вызывать из других фрагментов программы, размещенных не в этом файле, не используйте приведенную выше форму объявления функции встроенной. Включите ее либо в файл заголовка, либо в отдельный файл с расширением .inl, который подключите ко всем файлам, в которых есть к ней обращение, при помощи директивы #include. При такой организации компилятор всегда сможет найти код реализации этой функции.
Еще один нюанс. В Visual C++ 6.0 компилятор настолько разумен, что иногда может и проигнорировать ключевое слово или размещение кода реализации внутри определения класса. Если вы все же считаете себя умнее, заставьте его считаться с собой, используя ключевое слово __forceinline. Но будьте осторожны — это единственное ключевое слово в Visual C++ 6.0, которое не поддерживается другими компиляторами.

Возможно, вы уже знакомы с другими преимуществами C++. Функции в общем случае требуют передачи меньшего количества аргументов. В других языках в функции работы со счетами (например, Withdraw()) пришлось бы передавать и аргумент баланса, который нужно изменить. Но в C++, поскольку Withdraw() является методом класса, она может обращаться к членам-переменным объекта этого класса самостоятельно и не нуждается в передаче специального аргумента. Это делает код программы более понятным, а значит, упрощает сопровождение программы.

Инициализация объектов

В программе на языке С переменная объявляется одним оператором:

```
int i;
```

При желании можно объединить в этом операторе объявление и инициализацию переменной:

```
int i = 3;
```

Для того чтобы программный объект, представляющий счет в банке, имел какой-то смысл, в нем должны быть определены, как минимум, два члена — customer_id и account_num. Что касается члена balance, то начинать нужно, скорее всего, со значения 0. А как и где задать исходные значения для первых двух? В языке С++ каждый объект имеет метод инициализации, называемый *конструктором*. Именно на него и возлагаются все заботы по инициализации членов-переменных. Конструктор отличается от прочих методов класса: во-первых, имя этого метода всегда совпадает с именем класса объекта, а во-вторых, он не имеет возвращаемого значения, даже типа void. Для нашего класса BankAccount конструктор может быть запрограммирован, как показано в листинге А.5.

Листинг А.5. Конструктор класса BankAccount

```
BankAccount::BankAccount(char* customer, char* account, float startbalance)
        {
                strcopy( customet_id, customer);
                strcopy( account_num, account);
                balance = startbalance;
        }
```



Функция strcopy() входит в состав библиотеки времени выполнения языка С, к которой можно обращаться и из программ, написанных на С и С++. Эта функция копирует строковые переменные. В листинге А.5 при помощи этой функции значения, переданные конструктору в качестве аргументов вызова, копируются в члены-переменные объекта.

Помимо разработки реализации конструктора, нужно также включить его описание в объявление класса:

```
BankAccount(char* customer, char* account, float startbalance);
```

Обратите внимание на то, что тип возвращаемого значения в описании конструктора *отсутствует*. Не нужно также указывать и имя класса в операторе области видимости, поскольку описание конструктора находится внутри объявления класса. Строка описания должна завершаться точкой с запятой.

Теперь можно объявить и инициализировать объект класса BankAccount оператором:

```
BankAccount account("AB123456", "11038-30", 100.00);
```

Перегрузка

Теперь представим следующую ситуацию. Приложение, которое разрабатывается для обработки банковских счетов, должно работать и с кредитными карточками, которые представлены объектами класса CreditCard. В этом классе также нужно иметь метод подсчета текущего остатка на счете GetBalance(). В программе на языке С все функции являются глобальными и соответственно не может существовать двух разных функций с одинаковыми

именами. А в программе на C++ разные классы могут иметь одноименные методы. Рассмотрим следующий фрагмент программного кода:

```
BankAccount account("AB123456", "11038-30", 100.00);
float accountbalance = account.GetBalance();
CreditCard card("AB123456", "4500 000 000 000", 1000.00);
card.GetBalance();
```

Во втором операторе вызывается метод BankAccount::GetBalance() (т.е. метод класса BankAccount), а в четвертом — метод CreditCard::GetBalance() (т.е. метод класса CreditCard). Фактически эти функции имеют разные имена. Приведенный пример демонстрирует один из вариантов *перегрузки* (overloading). Перегрузка позволяет использовать простые и понятные имена для функций аналогичного назначения и избавиться от таких длинных имен, как GetBankAccountBalance() и GetCreditCardBalance().

Но еще более интересный эффект можно получить, используя перегрузку в пределах одного класса, т.е. можно использовать в одном и том же классе два и более методов с одинаковыми именами. В качестве примера рассмотрим уже упоминавшийся выше конструктор класса BankAccount. Совершенно очевидно, что при создании большинства объектов исходное значение остатка на счете должно быть равно 0. Поэтому невольно возникает мысль — а нельзя ли иметь в составе класса два конструктора. Первый должен принимать в качестве аргументов полный набор параметров — идентификатор клиента, номер счета и начальный баланс, а второй — только идентификатор клиента и номер счета. Это можно сделать, описав в объявлении класса два конструктора:

```
BankAccount(char* customer, char* account, float startbalance);
BankAccount(char* customer, char* account);
```

Как видно из листинга А.6, коды реализации обоих конструкторов очень похожи. Компилятор различает их по набору аргументов. Все описанное выше справедливо не только для конструкторов, но и для любых других функций — как методов классов, так и не входящих в состав какого-либо класса. Можно объявить две или более функции с одинаковыми именами, но с разным набором аргументов. Отличие в наборе может быть либо по количеству аргументов, либо по их типам.



Перегрузка не произойдет, если две функции с идентичными именами различаются *только* по типу *возвращаемого значения*.

Листинг А.6. Два конструктора класса BankAccount

```
BankAccount::BankAccount(char* customer, char* account, float startbalance)
      {
            strcopy( customet_id, customer);
            strcopy( account_num, account);
            balance = startbalance;
      }
BankAccount::BankAccount(char* customer, char* account)
      {
            strcopy( customet_id, customer);
            strcopy( account_num, account);
            balance = 0;
      }
```

Повторное использование кода и наследование

Объектно-ориентированный подход в программировании имеет еще одну очень привлекательную черту — возможность повторного использования уже разработанного кода. Во-первых, можно использовать классы, созданные другими разработчиками, не заботясь о деталях реализации методов внутри класса. Примером могут служить сотни классов из библиотеки MFC. Но более существенные преимущества дает повторное использование кода, реализуемое через механизм *наследования*.

Что такое наследование

Вновь вернемся к нашему примеру с банковскими счетами. Предположим, что после завершения отладки и тестирования класса BankAccount появилась идея добавить в систему еще и объекты для выполнения операций со срочными вкладами и чековыми книжками. Желательно, конечно, воспользоваться результатами уже выполненной работы — кодом, разработанным для класса BankAccount. Можно было бы просто скопировать подходящие фрагменты в реализацию новых классов, но язык C++ предоставляет возможность более изящного решения. Для начала задайтесь следующими вопросами.

Является ли чековый счет также и банковским счетом?

Является ли счет срочного вклада также и банковским счетом?

Если ответы на эти вопросы положительны, то можно создать соответствующие классы, используя иаследование класса BankAccount. В листинге А.7 представлено объявление двух новых классов.



Класс, использующий код другого класса, называется *производным* (derived) или *подклассом* (subclass). Класс, который предоставляет свой код другому классу, называется *базовым* (base) или *суперклассом* (superclass).

Листинг А.7. Объявление классов

```
class SavingsAccount: BankAccount
{
        private:
                float interestrate;

        public:
                SavingsAccount (char* customer, char* account, float startbalance,
float interest);
                void CreditInterest(int days);
};
class CheckingAccount: BankAccount
{
        public:
                CheckingAccount (char* customer, char* account, float startbalance);
                void PrintStatement (int month);
};
```

Теперь после создания объекта класса CheckingAccount можно вызывать для него как методы, унаследованные от класса BankAccount, так и новые методы, специфичные только для CheckingAccount, например:

```
CheckingAccount ca ("AB123456", "11038-30", 100.00);
ca.Deposit(100);
ca. PrintStatement(5);
```

Допустим, через некоторое время понадобилось изменить в программе технологию обслуживания клиентов. Кому-то пришла в голову идея брать дополнительную фиксированную плату с клиентов за выполнение операций занесения на счет и выдачи со счета (пусть это будет 10 центов). Эта плата должна накапливаться за некоторый промежуток времени (скажем, за месяц), а затем в конце месяца вычитаться из остатка на счете. Что придется в этом случае изменить в программе? Вы открываете файл BankAccount.h и добавляете защищеный член-переменную servicecharges в определение класса BankAccount. В конструкторе устанавливается исходное значение 0 для этой переменной. В реализации методов Deposit() и Withdraw() значение servicecharges увеличивается на 0.1. Затем в определение класса добавляется новый открытый метод ApplyServiceCharges(), который уменьшает значение остатка на счете на величину servicecharges, а последнюю сбрасывает в 0.

Все внесенные изменения автоматически будут распространены и на классы SavingsAccount и CheckingAccount.

Защита доступа

Еще только приступая к разработке метода CheckingAccount::PrintStatement(), можно предположить, что придется обратиться к члену-переменной, который хранит остаток на счете. Положим, это будет выполнено в виде оператора:

```
float bal = balance;
```

Но при попытке оттранслировать эту строку в составе кода CheckingAccount::PrintStatement() компилятор выдаст сообщение об ошибке. Это произойдет по той простой причине, что для члена-переменной balance в классе BankAccount установлен квалификатор доступа private, а значит, доступ к ней открыт только для членов данного класса. Хотя это может показаться и излишней предосторожностью, но выше уже был описан вариант модификации класса с заменой типа переменной, где эта предосторожность оказалась отнюдь не лишней. А что же тогда делать методу ? Для него есть метод, который приведет к желаемому результату.

Если же все-таки есть необходимость предоставить методам порожденного класса возможность прямого доступа к члену-переменной базового класса, вместо квалификатора доступа private можно использовать квалификатор protected. В этом случае определение класса будет иметь следующий вид:

```
class BankAccount
{
        protected:
                float balance;
        private:
                char[8] customer_id;
                char[8] account_num;

        // ...
};
```

Перегрузка методов

Может оказаться, что методы базового класса, унаследованные порожденным классом, не совсем соответствуют специфике последнего и нуждаются в определенной модификации.

Например, если класс BankAccount имеет метод Display(), который выводит на экран значения членов-переменных balance, customer_id и account_num, то в классе SavingsAccount этот метод должен, помимо отображения значений данных членов, выводить еще и значение interestrate. Для этого потребуется разработать отдельный метод SavingsAccount::Display(). Это и есть *перегрузка методов* базового класса.

Если желательно, чтобы метод SavingsAccount::Display() выполнял то же, что и метод базового класса, а затем и еще нечто, то наилучшим способом достичь такого результата будет вызов BankAccount::Display() внутри SavingsAccount::Display(). Но при этом нужно использовать полное имя метода базового класса, т.е. именно SavingsAccount::Display().

Полиморфизм

Полиморфизм есть одно из базовых понятий концепции объектно-ориентированного программирования. С ним встречаешься всякий раз при использовании механизма наследования и указателей на объекты производных классов. В качестве примера рассмотрим фрагмент программы в листинге А.8.

Листинг А.8. Наследование и указатели

```
BankAccount ba ("AB123456", "11038-30", 100.00);
CheckingAccount ca ("AB123456", "11038-32", 120.00);
SavingsAccount sa ("AB123456", "11038-39", 1000.00);

BankAccount* pb = &ba;
CheckingAccount* pc = &ca;
SavingsAccount* ps = &sa;

pb->Display();
pc->Display();
ps->Display();

BankAccount* pc2 = &ca;
BankAccount* ps2 = &sa;

pc2->Display();
ps2->Display();
```

В этом примере есть три объекта и пять указателей. Указатели pb, pc и ps совершенно очевидны и не требуют никаких дополнительных комментариев, а вот pc2 и ps2 часто называются указателями, приводящими к базовому классу (хотя в качестве значения для них задаются и адреса объектов производных классов, эти адреса приводятся к типу базового класса). Часто бывает удобно указатели на объекты разных производных классов объединить в единый массив однотипных указателей. В качестве единого типа может быть выбран только тип указателя на объекты базового класса, в данном случае — BankAccount. Затем такой массив может быть передан в качестве аргумента какой-либо функции, которая будет их обрабатывать в цикле. Эта функция не будет даже знать, что одни указатели имеют значения адресов объектов класса CheckingAccount, а другие — класса SavingsAccount.

А теперь проанализируем, что же произойдет при выполнении фрагмента программы из листинга А.8. Вызов pb->Display() будет передан функции BankAccount::Display(). Вызов pc->Display() будет передан функции CheckingAccount::Display(), если существует отдельная реализация для этой функции, но, поскольку таковая в нашем случае отсутствует, также будет вызвана функция базового класса BankAccount::Display(). Вызов ps->Display() будет передан перегруженному методу SavingsAccount::Display(). Все выполнено именно так, как

и задумано. Информация о каждом счете будет выведена подходящей именно для этого счета функцией.

Не так все просто в случае с указателями pc2 и ps2. Хотя один из них указывает на объект класса CheckingAccount, а другой — класса SavingsAccount, оба они имеют тип BankAccount*, т.е. указателя на объект класса BankAccount. По этой причине при каждом вызове произойдет обращенне к методу класса BankAccount::Display(), а это не совсем то, чего хотелось бы добиться. Чтобы достичь желаемого поведения программы, следует в объявление метода BankAccount::Display() добавить ключевое слово virtual, т.е. объявить его *виртуальным*. В результате будет приведен в действие механизм *полиморфизма*. Один и тот же оператор в коде программы будет выполнять разные операции в зависимости от того, на объект какого класса указывает переменная-указатель. Рассмотрим такой фрагмент программы:

```
void SomeClass::DisplayAccounts(BankAccount* a[], int numacconts)
{
        for (int i=0; i<numacconts; i++)
                a[i]->Display();
}
```

Эта функция получает в качестве аргумента массив указателей на объекты класса BankAccount и в цикле вызывает метод Display() для вывода на экран информации по каждому счету. Если, например, первый указатель в массиве относится к объекту класса CheckingAccount, будет вызван метод BankAccount::Display(). Если второй указатель в массиве относится к объекту класса SavingsAccount, а метод Display() объявлен в базовом классе как виртуальный, будет вызван SavingsAccount::Display(). Глядя на текст программы, нельзя заранее сказать, какой из методов будет вызван на каждой итерации цикла — все зависит от конкретных значений указателей в переданном функции массиве.

Это очень полезное свойство объектно-ориентированного программирования. Без него пришлось бы строить конструкцию switch...case и каким-то образом выяснять, объект какого класса адресуется каждым очередным элементом массива указателей. Кроме того, при появлении новых классов, производных от BankAccount и имеющих свои особенности отображения, пришлось бы вносить изменения и в функцию SomeClass::DisplayAccounts().

Управление памятью

Если объект объявляется в некотором блоке программы, он существует только до тех пор, пока выполняется этот блок. Затем объект выходит из области видимости программы и память, занятая им, возвращается операционной системе. Если желательно, чтобы при этом выполнялись еще какие-то операции, нужно разработать специальный метод (*деструктор*), который будет вызываться при уничтожении объекта, но до того, как занятая им память возвратится системе.

Иногда нужно сформировать объект, который должен существовать и после выполнения функции, в которой он был создан. Конечно, при этом нужно где-то сохранить указатель на объект.

Выделение и возврат памяти

В программе на языке С память выделяется функцией malloc(). Например, чтобы выделить память для хранения переменной типа, нужно использовать оператор:

```
int *pi = (int *) malloc( sizeof( int));
```

Но при создании объекта в C++ используется конструктор. А функция malloc(), разработанная задолго до появления на свет C++, понятия не имеет о конструкторах. В этом языке для выделения памяти используется новый оператор new, который не только выделяет память, но и вызывает соответствующий конструктор:

```
BankAccount* pb = new BankAccount("AB123456", "11038-30", 100.00);
```

Параметры, указанные после имени класса, передаются конструктору. Оператор не нуждается в специальных аргументах, указывающих объем выделяемой для объекта памяти.



При использовании оператора new память выделяется из *свободного запаса* (free store), который часто еще называют *кучей* (heap). А вот в случае, когда объект создается путем объявления в пределах блока, память для него берется из *стека*.

Когда необходимость в объекте, созданном оператором new, отпадет, освободиться от него можно с помощью оператора delete:

```
delete pb;
```

Оператор delete вызывает деструктор, а затем освобождает память и возвращает ее в распоряжение операционной системы. В программах на языке C эта функция возлагалась на free(), которую в новых программах на C++ ни в коем случае нельзя использовать для высвобождения памяти, выделенной оператором new. И в то же время, если уж вы используете где-либо в программе функцию malloc(), высвобождение соответствующей памяти должно производиться только функцией free() (ни в коем случае не delete). Многие разработчики вообще предпочитают навеки забыть даже о существовании malloc() и free(), а пользуются исключительно new и delete.

Оператор new можно использовать и для выделения памяти массиву:

```
int *numbers = new int[100];
```

Когда этот массив больше не нужен, выделенная для него память освобождается специальной формой оператора delete:

```
delete[] numbers;
```

Указатель в качестве члена-переменной

Использование указателей — это обычная практика в программировании на языках C и C++. Вернемся снова к нашему классу BankAccount. Почему бы не использовать вместо строкового идентификатора клиента указатель на объект класса Customer. Сделать это довольно просто — удалите в определении класса строку объявления переменной customer_id и добавьте указатель на объект класса Customer.

```
Customer* pCustomer;
```

Но учтите, что при этом нужно будет модифицировать конструктор, чтобы он находил нужный объект Customer по заданному идентификатору клиента. Возможно, вы сочтете целесообразным добавить еще пару конструкторов, в которых в качестве аргумента используется указатель на объект Customer. Но главное — не удаляйте ни одного метода из уже имеющихся в определении класса. Ведь в других модулях программы к ним есть обращения, и удаление любого из них может привести к необходимости пересмотра всего приложения.

Теперь все, что относится сугубо к клиенту (но не к его деньгам), можно передать классу Customer. Например, пусть этот класс занимается распечаткой сведений о клиенте — фамилии и адреса — на бланке чека:

```
pCustomer->PrintNameandAddress();
```

В дальнейшем все изменения формата распечатки сведений о клиенте будут касаться только класса Customer.



Такой способ повторного использования программного кода называется *агрегатированием* (aggregation), в отличие от наследования.

Динамические объекты

Каждый объект класса BankAccount всегда связан с соответствующим объектом класса Customer. Но возможен случай, когда к объектам BankAccount выборочно подключаются объекты другого класса, например класса CreditCard.

Чтобы реализовать такую возможность в классе BankAccount, нужно добавить еще один член — указатель на объект класса CreditCard:

```
CreditCard *pCard;
```

Во всех конструкторах нужно будет теперь установить исходное значение NULL этого указателя с тем, чтобы было ясно, что в исходном состоянии счет не связан ни с каким объектом описания кредитной карточки:

```
pCard = NULL;
```

Кроме того, нужно включить в состав класса открытый метод AddCreditCard(), который будет устанавливать значение для этого указателя. Такой метод можно реализовать как встроенный:

```
void AddCreditCard(CreditCard* card) {pCard = card;}
```

Модифицированный метод Withdraw() может выглядеть следующим образом:

```
void BankAccount::WithDraw(float ammounttowithdraw)
	{
		balance -= ammounttowithdraw;
		if( balance < 0)
		{
			if (pCard)
			{
				int hundreds = -(int) (balance / 100);
				hundreds++;
				pCard->CashAdvace(hundreds * 100);
				balance += hundreds * 100;
			}
			else
				balance += ammounttowithdraw; // Отменить снятие со
счета.
		}
	}
```

Деструкторы и указатели

Если объект модифицированного класса BankAccount уничтожается, то связанные с ним объекты классов Customer и CreditCard продолжают существовать. Это означает, что класс BankAccount нуждается в деструкторе. Вам не раз придется столкнуться с такой ситуацией при работе с классами, в которых указатели используются в качестве членов-переменных.

Рассмотрим ситуацию, которая возникает, когда выписывается новый чек. Когда чек будет предъявлен к погашению, израсходованную сумму необходимо будет вычесть из остатка на счете. А до тех пор информацию о нем целесообразно хранить в объекте класса CheckOrder. Скорее всего, для создания такого объекта при выписке чека в класс CheckingAccount придется включить метод OrderCheck(). Не вдаваясь в подробности, положим, что код этого метода будет выглядеть примерно так:

```
CheckingAccount:: OrderCheck()
{
	pOrder = new CheckOrder( /* аргументы конструктора */);
}
```

Очевидно, в состав защищенных членов-переменных класса CheckingAccount нужно будет добавить новый — указатель на объект класса CheckOrder:

```
CheckOrder* pOrder;
```

В конструкторе класса CheckingAccount этой переменной нужно присвоить значение NULL, поскольку при открытии счета чек автоматически не выписывается и соответственно объект CheckOrder, связанный со счетом, не существует.

Когда чек предъявляется к погашению, израсходованная сумма снимается с остатка на счете, что выполняется методом ChecksArrive() :

```
CheckingAccount::ChecksArrive()
{
	balance -= pOrder.GetCharge();
	delete pOrder;
	pOrder = NULL;
}
```

На заметку Обращаться непосредственно к члену balance базового класса, как это сделано в приведенном фрагменте программы, можно только а том случае, если он объялен с квалификатором доступа protected, а не private.

Оператор запустит деструктор объекта чека, и в результате будет высвобождена занятая им память, а присвоение указателю значения NULL предотвратит любые попытки обратиться к уже несуществующему объекту CheckOrder.

А что произойдет, если уничтожить объект класса CheckingAccount в то время, когда связанный с ним объект CheckOrder еще существует? Если при этом не обратить внимания на значение указателя pOrder, то объект останется в памяти, но доступ к нему будет навеки утерян. Поэтому деструктор класса должен позаботиться и об уничтожении всех объектов, на которые указывают члены-переменные головного объекта.

Именно в деструкторе должны уничтожаться связанные объекты, если они создаются в процессе выполнения методов класса. Деструктор всегда имеет отличительный признак — символ “тильда” (~) перед именем, которое совпадает с именем класса. Деструктор в отличие от остальных методов не имеет аргументов. Например, деструктор класса CheckingAccount будет выглядеть следующим образом:

```
CheckingAccount::~CheckingAccount()
{
	delete pOrder;
}
```

Непреднамеренный запуск деструктора

Если деструктор класса выполняет уничтожение и связанных объектов, нужио учитывать побочные эффекты, возникающие при работе с объектами класса. Рассмотрим код в листинге А.9. В нем создается объект класса CheckingAccount, выписывается чек, объект передается в качестве аргумента какой-то сторонней функции, а затем чек погашается.

Листинг А.9. Уничтожение объекта как результат побочного эффекта

```
CheckingAccount ca("AB123456", "11038-32", 200.00);
ca.OrderCheck();
SomeFunction(ca);
ca.ChecksArrive();
```

На первый взгляд, в тексте программы нет никаких причин для беспокойства. Однако, когда объект CheckingAccount передается функции SomeFunction(), система формирует копию этого объекта и именно ее передает функции. Объект-копия абсолютно идентичен объекту ca. Указатель в нем адресуется к тому же самому объекту CheckOrder, что и ca. Когда выполнение функции SomeFunction() будет завершено, объект-копия должен быть уничтожен, для чего система автоматически вызовет соответствующий деструктор. Последний же, ничуть не сомневаясь, уничтожит и объект CheckOrder, как запрограммировано в деструкторе. В результате метод ChecksArrive() натворит невесть чего, поскольку место в памяти, где полагается быть объекту счета, уже занято операционной системой и что она туда поместила, не знает даже Билл Гейтс. По закону бутерброда программа обязательно снимет со счета пару миллионов долларов вместо потраченных десяти. Так что плакали ваши денежки...

Как же поступить в такой ситуации? Во-первых, нужно так запрограммировать функцию SomeFunction(), чтобы она имела дело не с объектом как таковым, а с *указателем* на него или чтобы объект передавался ей *по ссылке*. Во-вторых, нужно разработать специальный *конструктор копирования*, который будет создавать временные копии объекта класса. Работа со ссылками и методика создания конструкторов копирования выходит за рамки этой книги, но главное, что вы должны почерпнуть из приведенного примера, — необходимость тщательно продумывать все побочные эффекты, возникающие при работе с временными копиями объектов классов.

Что еще следует знать

При более серьезном изучении языка С++ вам обязательно нужно познакомиться с множеством других средств этого языка. Некоторые из них перечислены ниже.

- Значения аргументов по умолчанию
- Оператор инициализации конструктора
- Ключевое слово const
- Передача параметров по ссылке
- Возврат значения по ссылке
- Статические члены-переменные и статические методы
- Конструктор копирования
- Перегрузка операторов

Еще две темы выходят за рамки ознакомительного курса — это исключения и шаблоны. Но с ними вы можете ознакомиться в главе 26 данной книги.

ПРИЛОЖЕНИЕ Б

Программирование для Windows и класс CWnd

В этом приложении...

Программирование для Windows

Инкапсуляция в Windows API

Содержимое класса CWnd

Дескрипторы классов MFC

Библиотека Microsoft Foundation Classes (MFC — базисные классы Microsoft) была написана с единственной целью — сделать программирование для Windows проще за счет включения в классы методов и данных, которые поддерживают задачи, общие для всех программ Windows. Классы в MFC разработаны специально для использования программистами в среде Windows. Методы внутри каждого класса решают задачи, в выполнении которых чаще всего нуждаются программисты в среде Windows. Многие классы имеют скрытое отношение к структурам и *классам окон* в старом понимании класса в Windows. Большинство методов относится непосредственно к функциям API (Application Programming Interface — программный интерфейс приложений), которые уже знакомы программистам, работающим в среде Windows.

Программирование для Windows

Программистам на языке C для Windows известно, что слово *класс* использовалось для описания окна задолго до начала программирования на C++ в Windows. Оконный класс характерен для любой программы, работающей в среде Windows C. Данные, описывающие этот класс, содержит стандартная структура и множество стандартных оконных классов поддерживается операционной системой. Программист обычно строит новый оконный класс для каждой программы и регистрирует его путем вызова функции API RegisterClass(). На базе этого класса могут быть созданы окна, появляющиеся на экране, с помощью другой функции API —CreateWindow().

Оконные классы в стиле языка C

Структура WNDCLASS, которая описывает оконный класс, эквивалентна структуре WNDCLASSA, приведенной в листинге Б.1.

Листинг Б.1. Структура WNDCLASSA из файла WINUSER.H

```
typedef struct tagWNDCLASSA {
   UINT       style;
   WNDPROC     lpfnWndProc;
   int        cbClsExtra;
   int        cbWndExtra;
   HINSTANCE   hInstance;
   HICON       hIcon;
   HCURSOR     hCursor;
   HBRUSH      hbrBackground;
   LPCSTR      lpszMenuName;
   LPCSTR      lpszClassName;
} WNDCLASSA, *PWNDCLASSA, NEAR *NPWNDCLASSA, FAR *LPWNDCLASSA;
```

В файле WINUSER.H установлены две очень похожие структуры оконных классов: WNDCLASSA для программ, использующих обычные строковые переменные, и WNDCLASSW для программ, использующих Unicode. Последние описаны в главе 28.



Файл WINUSER.H поставляется в составе Visual Studio. Он обычно находится в папке \Program Files\Microsoft Visual Studio\VC98\include.

При разработке C-программ для Windows необходимо заполнять структуру WNDCLASS, включающую следующие компоненты.

- style — число, получаемое путем комбинирования стандартного стиля, который представлен константами вида CS_GLJBALCLASS или CS_OWNDC, с битовым терминальным оператором OR (|). Вполне нормально зарегистрировать класс со значением стиля, равным 0; другие стили предусмотрены для специальных окон.
- lpfnWndProc — указатель на функцию типа Windows Procedure (Оконная процедура), называемую обычно WndProc, для данного класса. Эта функция обсуждается в главе 3.
- cbClsExtra — параметр, который определяет, сколько дополнительной памяти (байтов) нужно добавить к оконному классу. Обычно значение этого параметра равно 0, но программисты, работающие на языке С, иногда строят оконные классы с дополнительными данными внутри этих классов.
- cbWndExtra — параметр, который определяет, сколько дополнительной памяти (слов) нужно добавить к оконному классу. Обычно значение этого параметра равно 0.
- hInstance — дескриптор для экземпляра приложения, выполняемая программа, которая регистрирует этот оконный класс. Теперь можно рассматривать его как способ, с помощью которого экземпляр оконного класса может определить использующее его приложение.
- hIcon — пиктограмма, которая должна быть нарисована после минимизации окна. Она, как правило, устанавливается путем обращения к другой функции API, LoadIcon().
- hCursor — отображает курсор, когда указатель мыши расположен в окне, ассоциированном с этим оконным классом. В типичных случаях параметр устанавливается путем обращения к функции API LoadCursor().
- hbrBackground — кисть, используемая для заполнения фона окна. Обычным способом установки этой переменной является вызов функции API GetStockObject().
- lpszMenuName — двойной указатель на строковую переменную (имя меню для оконного класса), равный NULL при прерывании (завершении) приложения.
- lpszClassName — имя оконного класса, используемого функцией CriateWindow() при создании окна (экземпляра оконного класса).

Создание окна

Программист, никогда не писавший программ в Windows, возможно, будет слегка напуган необходимостью заполнения структуры WNDCLASS. Но это — первый шаг в программировании на языке С в Windows. Тем не менее всегда можно найти такой простой пример программ для копирования, как, например, этот:

```
WNDCLASS wcInit;
        wcInit.style = 0;
        wcInit.lpfnWndProc = (WNDPROC)MainWndProc;
        wcInit.cbClsExtra = 0;
        wcInit.cbWndExtra = 0;
        wcInit.hInstance = hInstance;
        wcInit.hIcon = LoadIcon (hInstance, MAKEINTRESOURCE(ID_ICON));
        wcInit.hCursor = LoadCursor (NULL,IDC_ARROW);
        wcInit.hbrBackground = GetStockObject (WHITE_BRUSH);
        wcInit.lpszMenuName = "DEMO";
        wcInit.lpszClassName = "NewWClass";
        return (RegisterClass (&wcInit));
```

Венгерская нотация

Пусть вас не удивляют такие, на первый взгляд, неудобочитаемые, имена переменных, как lpszClassName и wcInit. Почему не просто Init? Программисты Microsoft для идентификации переменных руководствуются негласным соглашением, известным как *венгерская нотация*. Она названа так потому, что ее в Microsoft популяризировал венгерский программист Чарльз Шимоньи (Charles Simonyi). И действительно, на первый взгляд кажется, будто идентификаторы переменных написаны не на английском, а на каком-то другом языке.

В венгерской нотации переменным даются описательные имена, такие как Count и ClassName, начинающиеся с заглавных букв. Если имя состоит из нескольких слов, каждое слово начинается с заглавной буквы. Затем перед описательным именем добавляются буквы, чтобы указать тип переменной. Например, nCount для типа int и bFlag для типа BOOLEAN (True False) переменных. Теперь программист не сможет забыть о типе переменной или сделать такие глупости, как передача функции, ожидающей значения без знака, числовой знаковой переменной.

Этот стиль приобрел широкую популярность, хотя некоторые программисты его ненавидят. Если вы еще по-прежнему за старые добрые времена, когда спорили о том, где поставить ограничительные скобки или, что еще интереснее, как их назвать (ограничительными, фигурными или квадратными), но не можете найти того, кто согласился бы возобновить старые споры, то при желании обязательно найдете и оппонента венгерской нотации. Аргументы в этом споре лежат между *Вы сами вынуждаете себя делать глупые ошибки* и *Это безобразно и трудно читаемо*. Но дело обстоит таким образом, что структуры, применяемые API, и классы, определенные в MFC, используют венгерскую нотацию, поэтому вам тоже придется пользоваться ею. Вполне возможно, что вы уже давно ею пользуетесь, но в несколько другом виде (и не знаете, что она *венгерская*, как герой Мольера, который и не догадывался, что всю жизнь говорил прозой). Все дело в соглашении о префиксах. В венгерской нотации предлагаются следующие префиксы соответственно типам переменных.

Префикс	Тип переменной	Комментарий
a	Массив	—
b	Булева	—
d	Число с двойной точностью	
h	Дескриптор	—
i	Целое число	Индекс в ...
lp	Двойной указатель	—
lpfn	Двойной указатель на функцию	—
m_	Переменная — член структуры или класса	—
n	Целое число	Количество чего-либо
p	Указатель	
s	Строковая переменная	
sz	Строковая переменная, ограниченная нулем	
u	Целое число без знака	
C	Класс	

Многие программисты добавляют собственные типовые префиксы к именам переменных, например wc в wcInin указывает на принадлежность оконному классу.

Заполнение структуры wcInit и вызов функции RegisterClass()является стандартным средством; в данном случае регистрируется класс NewClass с меню DEMO и функцией обработки WindProc(), названной MainWndProc().

После регистрации класса программа формирования окна, созданная опытным программистом, не один год покоряющим Windows, может выглядеть так:

```
HWND hWnd;
hInst = hInstance;
hWnd = CreateWindow (
        "NewWClass",
        "Demo 1",
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT,
```

```
        CW_USEDEFAULT,
        CW_USEDEFAULT,
        CW_USEDEFAULT,
        NULL,
        NULL,
        hInstance,
        NULL);
if (! hWnd)
        return (FALSE);
ShowWindow (hWnd, nCmdShow);
UpdateWindow (hWnd);
```

Эта программа последовательно вызывает функции CreateWindow(), ShowWindow() и UpdateWindow(). Параметры функции API CreateWindow() приведены ниже.

- lpClassName — указатель на имя класса, который использовался при вызове функции RegisterClass().
- lpWindowName — имя окна.
- dwStyle — стиль окна, который формируется путем объединения с помощью терминального оператора поразрядного ИЛИ (|) констант, определенных директивой #define. Для простейшего окна приложения, как в примере, параметр WS_OVERLAPPEDWINDOW является стандартным.
- x — позиция окна по горизонтали. Параметр CW_USEDEFAULT позволяет операционной системе на основе установок экрана вычислять значения параметров по умолчанию.
- y — позиция окна по вертикали. Параметр CW_USEDEFAULT позволяет операционной системе на основе установок экрана вычислять значения параметров по умолчанию.
- nWidth — ширина окна. Параметр CW_USEDEFAULT позволяет операционной системе на основе установок экрана вычислять значения параметров по умолчанию.
- nHeight — высота окна. Параметр CW_USEDEFAULT позволяет операционной системе на основе установок экрана вычислять значения параметров по умолчанию.
- hWndParent — дескриптор родительского окна или окна, которое по отношению к данному выступает в роли владельца. (Некоторые окна создаются другими окнами; в этом случае одно из них является владельцем, а другое — собственностью. Другой вариант терминологии: первое окно — родитель, второе — потомок.) Значение NULL означает отсутствие владельца для этого окна.
- hMenu — дескриптор меню или идентификатор окна-потомка; другими словами, окно, принадлежащее этому же окну. NULL означает отсутствие потомка для этого окна.
- hInstance — дескриптор приложения, создающего это окно.
- lpParam — указатель на любые дополнительные параметры. В приведенном выше примере ни один из них не понадобился.

Функция CreateWindow()возвращает дескриптор окна — каждый пользователь запрашивает дескриптор своего окна hWnd, который потом применяется в остальной части программы. Если он равен NULL, окно не создано. Если возвращено значение дескриптора, отличное от NULL, окно сформировано, и дескриптор передается функциям ShowWindow()и UpdateWindow(). Совместными усилиями эти функции чертят на экране активное окно.

Дескрипторы

Дескриптор — это больше чем просто указатель. Программы для Windows с помощью дескриптора ссылаются на ресурсы наподобие окон, пиктограмм, курсоров и т.д. В системе Windows существует невидимая для разработчика таблица дескрипторов, которая отслеживает адрес ресурса и информацию о типе ресурса. Этот элемент данных назван *дескриптором (handle)* из-за того, что программа использует его как способ чтения содержимого (get hold of) ресурса. Дескрипторы передаются функциям, которые требуют использования ресурсов, и возвращаются функциями, распределяющими ресурсы.

Существует несколько типов дескрипторов: hWnd — дескриптор для окон, HICON — дескриптор для пиктограмм и т.д. Не имеет значения вид используемого дескриптора. Чтобы можно было его применять, нужно только запомнить способ доступа к ресурсу.

Инкапсуляция в Windows API

Функции API формируют окна и манипулируют ими на экране, объединяют программы с файлом справки, распределяют память и выполняют много других функций. Когда эти функции инкапсулированы в классы MFC, программы могут выполнять те же основные задачи Windows с меньшей долей участия программиста во всем этом процессе.

Функций API насчитывается буквально тысячи, а потому для получения полного представления об API придется потратить от шести месяцев до года. Мы не ставили перед собой задачи создания в рамках этой книги краткого руководства по API. Документация по функциям API доступна из Visual C++ — щелкните на вкладке InfoViewer в окне Workspace и раскройте тему Platform SDK (платформа SDK). В ней раскройте темы Reference (Ссылки), Functions (Функции) и Win32 Functions (функции Win32) для отображения в алфавитном порядке списка категорий, например, ArrangeIconicWindows (Организация пиктограмм в Windows) или CloseClipboard (Закрытие буфера обмена Clipboard). Внутри этих категорий функции расположены в алфавитном порядке. Имеется также возможность поиска определенных функций по индексам.

В предыдущем разделе уже говорилось о двух функциях API: RegisterClass()и CreateWindow(). Они дают хорошее представление о трудностях и проблемах, связанных с программированием на языке С для Windows с API, и о том, как классы MFC в значительной мере их устраняют.

Содержимое класса CWnd

Класс CWnd занимает исключительно важное место в MFC. В качестве базового его использует примерно третья часть всех классов MFC — такие классы, как CDialog, CEditView, CButton и многие другие. Он служит связующим звеном между старыми оконными классами и функциями API, которые формируют и обрабатывают оконные классы. Например, единственной открытой переменной-членом, хранящей дескриптор окна, является переменная m_hWnd. Эта переменная устанавливается функцией-членом CWnd::Create()и используется почти всеми функциями-членами, когда они вызывают свои ассоциированные функции API.

Резонно предположить, что вызов функции API CreateWindow()будет автоматически поддержан в конструкторе класса CWnd, CWnd::CWnd() и при вызове конструктора для инициализации объекта класса CWnd на экране сформируется соответствующее окно. Это избавило бы программиста от значительной части работы, поскольку уж конструктор-то никто не забудет вызвать! Однако Microsoft предпочла не вызывать CreateWindow()в конструкторе. Конструктор выглядит так:

```
CWnd::CWnd()
        {
                AFX_ZERO_INIT_OBJECT(CCmdTarget);
        }
```

AFX_ZERO_INIT_OBJECT — это макрос, который расширяется препроцессором компилятора C++ и использует функцию C memset для обнуления всех членов-переменных объекта, например:

```
#define AFX_ZERO_INIT_OBJECT(base_class) memset((base_class*)this)+1, 0,
sizeof(*this)
↳- sizeof(class base_class));
```

Причина, по которой Microsoft предпочитает не вызывать CreateWindow()в конструкторе, состоит в том, что конструктор не может возвращать значение переменной. Если при формировании окна что-то произошло, то не существует простого способа исправить ошибку. Конструктор почти ничего не делает такого, чего нельзя было бы исправить. Вызов функции CreateWindow()осуществляется внутри функции-члена CWnd::Create() или тесно с ней связанной функции CWnd::CreateEx(), которая выглядит, как в листинге Б.2.

Листинг Б.2. Функция CWnd::CreateEx()из файла WNDCORE.CPP

```
BOOl CWnd::CreateEx(DWORD dwExStyle, LPCTSTR lpszClassName,
        LPCTSTR lpszWindowName, DWORD dwStyle, int x, int y,
        int nWidth, int nHight, HWND hWndParent,
        HMENU nIDorHMENU, LPVOID lpParam)
{
// Допустима модификация общих параметров создания обьекта.
CREATESTRUCT cs;
cs.dwExStyle= dwExStyle;
cs.lpszClass= lpszClassName;
cs.lpszName= lpszWindowName;
cs.style= dwStyle;
cs.x=x;
cs.y=y;
cs.cx= nWidth;
cs.cy= nHight;
cs.hwndParent= hWndParent;
cs.hMenu= nIDorHMENU;
cs.hInstance= AfxGetInstanceHandle();
cs.lpCreateParams= lpParam;

if(!PreCreateWindow(cs)
        {
        PostNcDestroy();
        return FALSE;
        }
AfxHookWindowCreate( this);
HWND hWnd =::CreateWindowEx(cs.dwExStyle, cs.lpszClass,
        cs.lpszName, cs.style, cs.x, cs.y, cs.cx, cs.cy,
        cs.hwndParent, cs.hMenu, cs.hInstance, cs.lpCreateParams);
#ifdef _DEBUG
        if (hWnd == NULL)
        {
                TRACE1("Warning: Window creation failed: [ccc]
                GetLastError returns 0x%8.8X\n",
                GetLastError();
        }
#endif
```

```
if (!AfxUnhookWindowCreate())
      PostNcDestroy();
// Очистить, если CreateWindowEx() не сработала.
if (hWnd == NULL)
      return FALSE;
ASSERT( hWnd == m_hWnd);
return TRUE;
}
```



Файл WINCODE.CPP поставляется в составе Visual Studio. Он обычно находится в папке \Program Files\Microsoft Visual Studio\VC98\mfc\src.

В этом примере устанавливаются члены структуры CREATESTRUCT, очень похожей на структуру WNDCLASS соответственно значениям аргументов, переданным функции CreateEx(). Перед анализом hWnd и возвратом она вызывает функции PreCreateWindow(), AfxHookWindowCreate(), ::CreateWindow() и AfxUnhookWindowCreate().



Префикс AFX, используемый для многих полезных функций MFC, возвращает нас к тем дням, когда внутренним именем для библиотеки классов Microsoft было Application Framework. Символ :: в операции вызова функции CreateWindow()идентифицирует ее как функцию API. В этом контексте иногда на нее ссылаются, как на функцию SDK. Другие функции являются функциями-членами CWnd, которые незаметно для программиста выполняют всю подготовительную работу.

Итак, учитывая изложенное, кажется, что запоминать что-либо нет необходимости. Вы создаете экземпляр класса CWnd, вызываете его функцию Create()и должны передать ей столько же параметров, сколько в старом варианте С. Так в чем же смысл всех нововведений? Действительно, класс CWnd является именно тем классом, который наследуется всеми остальными. Ситуация значительно упрощается в порожденных классах. Например, рассмотрим Cbutton — класс, который инкапсулирует функции кнопки в диалоговом окне. Кнопка — это не что иное как маленькое окно, но ее возможности ограничены, например пользователь не может изменить размер кнопки. Ее метод Create выглядит так:

```
BOOL CButton::Create(LPCTSTR lpszCaption, DWORD dwStyle, const RECT& rect, CWnd*
pParentWnd, UInt nID)
{
      CWnd* pWnd = this;
      return pWnd.>Create(_T("BUTTON"), lpszCaption, dwStyle, rect, pParent, nID);
}
```

Некоторые из них имеют еще меньше аргументов. Если вам нужна кнопка, вы ее создаете и позволяете классу заполнять остальные параметры, предусмотренные другими уровнями иерархии.

Дескрипторы классов MFC

Существует более 200 классов MFC. Почему так много? И зачем они нужны? Как может нормальный человек их запомнить и знать назначение каждого класса? Спросите что-нибудь полегче. Ответы на эти вопросы займут большую часть данной книги. В первой ее части представлены чаще всего используемые классы MFC. В этом же разделе мы рассмотрим несколько наиболее важных *базовых* классов.

Класс CObject

На рис. Б.1 приведена схема верхнего уровня дерева наследования для классов в библиотеке MFC. Редкие классы из библиотеки MFC не являются наследниками класса CObject. Он наделен основными функциональными возможностями, необходимыми для всех классов MFC, — поддержкой *сохранения-восстановления* и выводом с диагностикой. Кроме того, классы, порожденные CObject, могут быть объединены в классы-контейнеры, обсуждаемые в приложении Е.

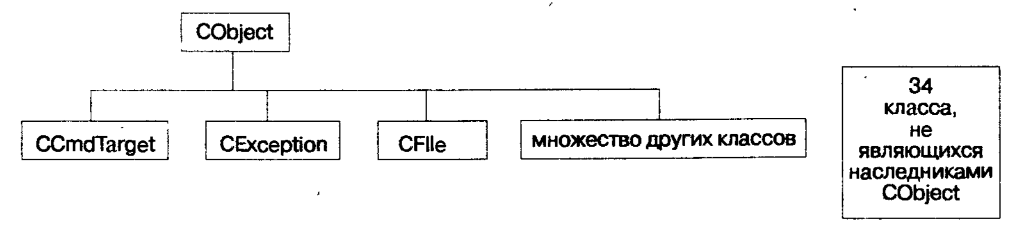


Рис. Б.1. Почти все классы в библиотеке MFC являются наследниками CObject

Класс CCmdTarget

Для некоторых производных от CObject классов, таких как CFile и Cexception, и порожденных ими классов нет необходимости организовывать непосредственный диалоговый обмен с пользователем и операционной системой через сообщения и команды. На рис. Б.2 приведены классы, порожденные CCmdTarget и обычно называемые *объектами воздействия команд* (command targets).

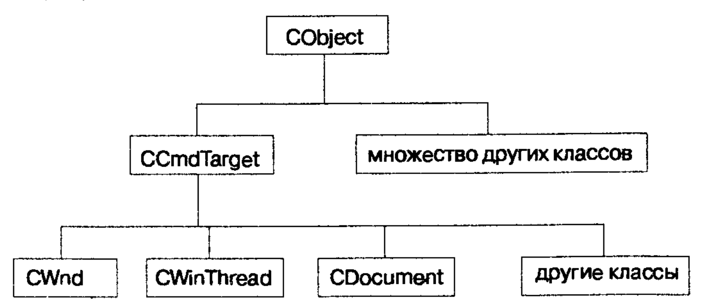


Рис. Б.2. Любой класс, получающий команды, должен быть наследником класса CCmdTarget

Класс CWnd

Как уже отмечалось, CWnd является исключительно важным классом. Только классы, порожденные CWnd, могут получить сообщение; *документы* и *потоки* (threads) могут получать команды, но не сообщения.



В главе 3 анализируется разница между командами и сообщениями, а понятие о документе в том смысле, в котором он используется в С-программах, разъясняется в главе 4. В главе 27 дается представление о потоках — параллельно выполняемых функциях.

Класс CWnd обеспечивает специфические для Windows процессы, в частности вызов функций CreateWindow()и DestroyWindow(). Методы этого класса поддерживают собственно формирование изображения окна на экране, обработку сообщений, диалог с буфером Clipboard и выполняют множество других операций. Общее количество методов класса CWnd — почти 250. Только несколько из этих функций потребуется перегрузить в порожденных классах. На рис. Б.3 показаны классы — наследники CWnd; так как классов элементов управления слишком миого для того, чтобы приводить их список, на рисунке они объединены.

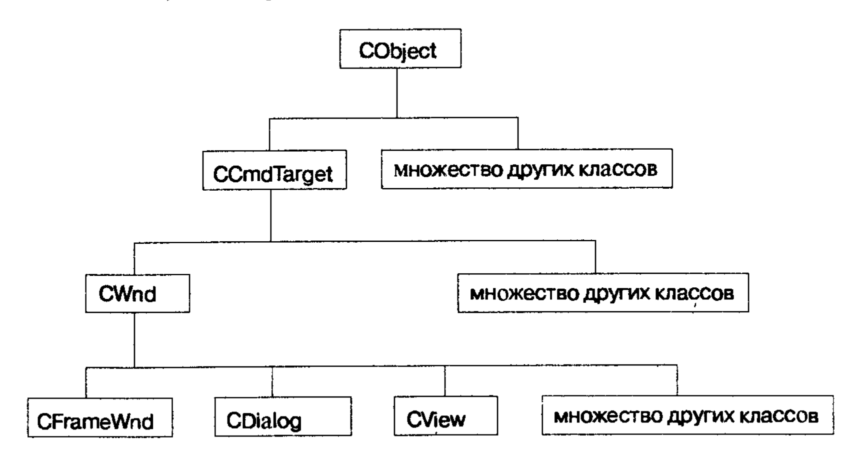


Рис. Б.3. Любой класс, который будет принимать сообщения, должен быть наследником класса CWnd

Остальные классы

На приведенных выше рисунках перечислены десять классов. Как быть с остальными двумястами классами? Вы узнаете о них из данной книги. Если это специальный класс, который вас удивил, поищите его в тематическом указателе. Обратитесь также к файлу справки, поскольку каждый класс в нем документирован. И не забывайте, что полный исходный текст для MFC включен во все копии Visual C++. Конечно, читая полный исходный текст, трудно понять, как работает класс, но иногда действительно необходимо более детально изучить какой-либо класс.

ПРИЛОЖЕНИЕ B

Интерфейс Visual Studio

В этом приложении...

Интегрированная среда разработки Visual Studio

Выбор средств просмотра

Просмотр элементов интерфейса

Просмотр текста программы, организованный соответственно классам

Просмотр файлов программ

Выходные сообщения и сообщения об ошибках

Редактирование текстов программ

Система меню

Интегрированная среда разработки Visual Studio

Приобретя Microsoft Visual C++, вы фактически получаете Microsoft Visual Studio с установленными компонентами Visual C++. Visual Studio — это значительно больше чем просто компилятор, поэтому изучать его необходимо тщательнее, чем может показаться на первый взгляд. Интерфейс Visual Studio очень наглядный, что облегчает знакомство с Visual C++.

Microsoft Visual C++ является одним из компонентов Microsoft Visual Studio. Возможности этого программного инструмента удивительны. Он не случайно получил название *интегрированной среды разработки* (Integrated Development Environment — IDE), так как, не выходя за рамки этой среды, можно решать целый комплекс задач.

- Формировать заготовки приложений без написания текстов программ
- Просматривать проект несколькими различными способами
- Редактировать файлы заголовков и текстов программ
- Формировать визуальный графический интерфейс приложения (меню и диалоговые окна)
- Компилировать и компоновать программы
- Отлаживать приложение в процессе его выполнения

Говоря техническим языком, Visual C++ — это один из компонентов Visual Studio. С одной стороны, можно купить, например, компилятор Microsoft Visual J++ и также успешно использовать его в среде Visual Studio. С другой стороны, Visual C++ — это больше чем просто компонент Visual Studio, поскольку библиотека MFC (*Microsoft Foundation Classes*), которая стала стандартом для программирования на языке C++ в среде Windows, является библиотекой классов и не связана со средой разработки. Большинство компиляторов C++ используют сейчас MFC. Тем не менее для большинства программистов Visual C++ и Visual Studio означает одно и то же, поэтому в нашей книге эти названия часто перемежаются.

Выбор средств просмотра

Интерфейс Visual Studio очень нагляден и хорошо помогает пользователю при просмотре компонентов проекта — ресурсов, классов, файлов или диалоговой документации. Основной экран разделен на ячейки, размеры которых можно менять по своему усмотрению. Для облегчения решения стандартных задач имеется много контекстных меню, доступ к которым осуществляется с помощью щелчков правой кнопкой мыши на различных компонентах изображения на экране.

С помощью Visual C++ можно работать с единственным приложением, как с проектом. *Проект* — это набор файлов: заголовков, текстов программ, ресурсов, установок, конфигураций. Visual Studio дает возможность работать со всеми компонентами проекта одновременно. Для работы с приложением необходимо открыть проект (файл с расширением `.dsw`), что проще, чем открывать независимо каждый файл программы. Интерфейс Visual Studio, приведенный на рис. В.1 и В.2, разработан в расчете на работу с несколькими компонентами проекта. Поэтому экран разделен на несколько зон (окон). Для того чтобы разработать новое приложение, нужно создать новый проект.

Рис. В.1. Интерфейс Visual Studio предоставляет пользователю большой объем информации. Окно Workspace (Компоненты проекта) находится слева

Рис. В.2. Когда окно Workspace сужено, от надписей на корешках вкладок остаются одни пиктограммы

Экран Visual Studio разделен на следующие зоны.

- Вверху — меню и панели инструментов. Они обсуждаются во второй половине этой главы.
- Слева — окно Workspace — компоненты проекта.
- Справа — основная рабочая область, в которой можно редактнровать файлы.
- Внизу — окно выходных данных и строка состояния.



Откройте Visual Studio и, следуя описанию различных функций среды в этой главе, изменяйте размеры соответствующих зон экрана.

Окно Workspace определяет, с каким компонентом проекта выполняется работа в текущий момент, и соответственно организует вывод информации в основную рабочую область — программные файлы или ресурсы (меню, пиктограммы и диалоговые окна). Подробно каждое из средств просмотра обсуждается в отдельных разделах главы.

- Вкладка ResourceView обсуждается в разделе *Просмотр элементов интерфейса.*
- Вкладка ClassView обсуждается в разделе *Просмотр текста программы, организованный соответственно классам.*
- Вкладка FileView обсуждается в разделе *Просмотр файлов программ.*

Visual Studio для отслеживания всей ннформации о проекте использует два файла. *Файл компонентов проекта (Project workspace file)* с расширением .dsw содержит имена файлов, включенных в состав проекта, каталогов, в которых они находятся, опции компилятора и компоновщика и другую информацию, необходимую для работы с проектом. Существует еще и *файл проекта (project file)* с расширением .dsp для каждого проекта в составе комплексного проекта. *Файл опций проекта (workspace option file)* с расширением .opt содержит все установки пользователя для Visual Studio — цвета, шрифты, панели инструментов, перечень открытых файлов, положение и размеры их MDI-окон, точки останова для отладки и т.д. Если кто-то из ваших коллег пожелает работать с вашим проектом, он должен обязательно иметь копии файла компонентов проекта и файла проекта, а вот файл опций проекта передавать необязательно.

Чтобы открыть проект, необходимо открыть файл компонентов проекта. Остальные файлы откроются автоматически.

Просмотр элементов интерфейса

Щелчок на вкладке ResourceView позволяет вывести в окне Workspace древовидный список визуальных элементов разрабатываемого приложения — акселераторы, диалоговые окна, пиктограммы, меню, таблицы строк и информацию о версии. Эти ресурсы определяют способ взаимодействия пользователей с разрабатываемой программой. В главах 2, 8 и 10 описаны методы и средства создания и редактирования этих ресурсов. В следующих нескольких разделах описываются способы просмотра завершенных ресурсов.



Откройте один из созданных ранее проектов и воспроизведите те операции, которые мы будем в дальнейшем описывать. Для этой цели вполне подойдет приложение ShowString из главы 8, так как в нем используется большинство функций, описываемых в этом разделе.

Акселераторы

Акселераторы связывают комбинации клавиш с пунктами меню. На рис. В.3 приведен список ресурса акселераторов, созданный AppWizard. При создании нового приложения он автоматически включает в проект все эти комбинации акселераторов. При необходимости для отдельных пунктов меню можно добавить горячие клавиши.

Рис. В.3. Акселераторы связывают комбинации клавиш с пунктами меню

Диалоговые окна

Диалоговые окна — это средство, которое используется приложением для приема информации от пользователя. Когда диалоговый ресурс выведен в основную рабочую область Visual Studio, как показано на рис. В.4, появляется плавающая панель инструментов — набора элементов управления. (Если она не появилась на экране, щелкните на строке меню правой кнопкой мыши и выберите Controls из контекстного меню.) Каждая пиктограмма панели представляет элемент управления (текстовое поле, список, кнопку и т.д.), который можно поместить в диалоговое окно приложения.

Окно Properties (см. рис. В.4) можно вывести на экран с помощью команды View⇨Properties (<Alt+Enter>). В нем можно настроить характеристики как отдельных элементов управления, так и всего диалогового окна.



Чтобы это окно оставалось на экране продолжительное время, щелкните на изображении канцелярской кнопки в левом верхнем углу окна Properties.

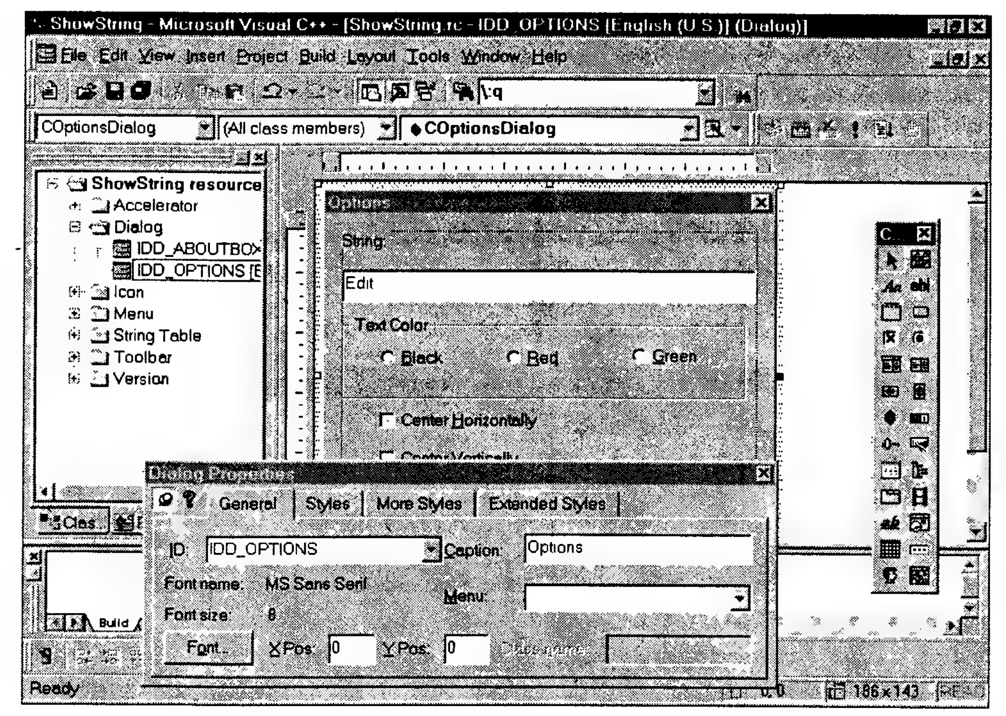


Рис. В.4. Создание диалогового окна в среде Visual Studio

Само появление названия *Visual C++* в определенной мере связано с методикой редактирования диалоговых окон. Если вы желаете сдвинуть кнопку левее в диалоговом окне, щелкните на ней мышью, перетащите ее на новое место и отпустите кнопку мыши. Аналогично, если необходимо изменить размеры диалогового окна, захватите угол или границу и сдвиньте в нужном направлении, как для любого другого окна с регулируемыми размерами. До разработки Visual C++ этот процесс сопровождался редактированием текста программ и подсчетом пикселей. Длился он минуты, если не часы, а не несколько секунд, как теперь. Такая методика построения диалоговых окон в среде Visual Studio пришлась весьма по душе разработчикам интерактивных приложений.

Пиктограммы

Пиктограммы — это маленькие растровые картинки, представляющие некоторую программу или ее документы. Например, пиктограмма используется для представления программы после ее минимизации. Вариант пиктограммы увеличенного размера используется для представления как программ, так и документов внутри окна программы `Explorer` (Проводник). Пиктограммой представляется и минимизированное окно документа MDI-приложения. На рис. В.5 приводится пиктограмма, предлагаемая `AppWizard` по умолчанию для минимизированного окна документа приложения такого типа. При создании любого приложения рекомендуется заменить ее вашей собственной, которая позволит нагляднее представить назначение программы.

Пиктограмма — это растровое изображение размером 32×32 пикселя, которое может редактироваться любым графическим редактором, в том числе и простейшим его вариантом, включенным в Visual Studio. Интерфейс этого редактора аналогичен интерфейсу `Microsoft Paint` и `Microsoft Paintbrush`. Можно строить изображение по одному пикселю, щелкая на них мышью, или наносить линии, щелкнув мышью и затем протянув "след". Можно формировать как малые, так и большие пиктограммы и в обоих случаях сразу видеть результат.

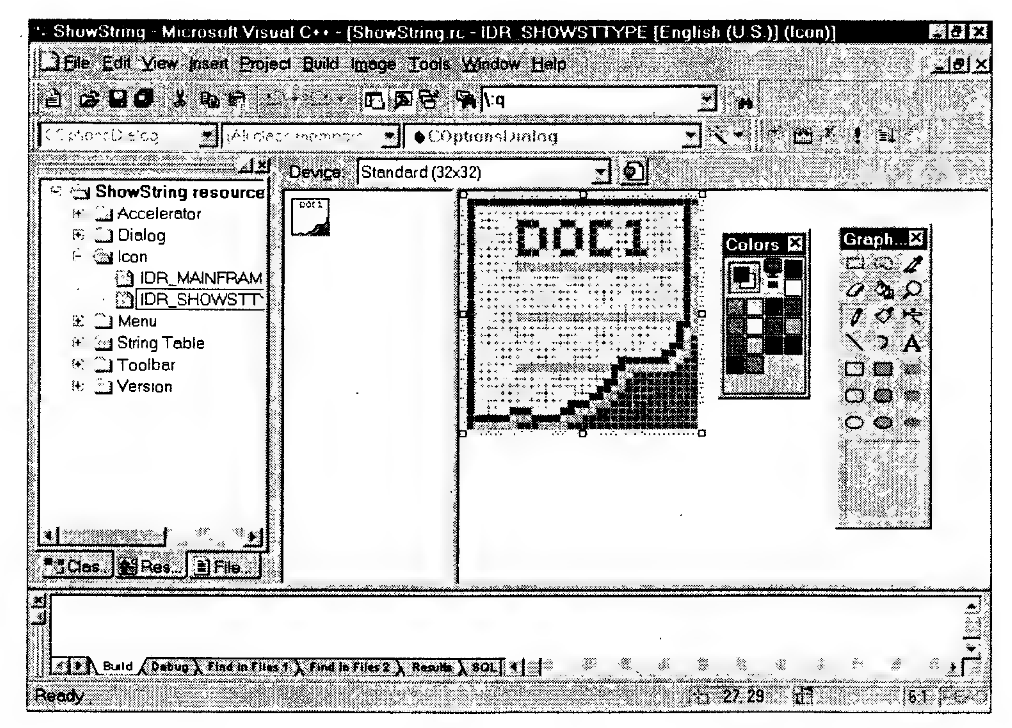


Рис. B.5. Пиктограммы представляют разрабатываемое приложение и его документы

Меню

Меню — это средство, с помощью которого пользователь указывает программе, что нужно делать. *Клавиши ускоренного вызова (акселераторы)* (Keyboard shortcuts — accelerators) связаны как с пунктами меню, так и с соответствующими пиктограммами панели инструментов. AppWizard формирует для нового приложения стандартные меню, которые затем можно редактировать. С помощью Class Wizard можно связать пункты меню с функциями в тексте программы. На рис. B.6 в окне ResourceView показано меню, выделенное в списке ресурсов окна компонентов проекта. Для пункта меню можно вызвать на экран диалоговое окно Properties. Для этого нужно выбрать View⇨Properties. Каждый пункт меню характеризуется следующими тремя параметрами.

- *Поле* ID *(Идентификатор ресурса)*. Этот параметр уникально идентифицирует конкретный пункт меню. С идентификаторами ресурса связаны акселератор и пиктограмма панели инструментов. Существует соглашение о выборе идентификатора для пункта меню, соблюдение которого позволяет довольно легко определить, к какому именно пункту и какого меню относится данный идентификатор. На рис. B.6 идентификатором ресурса является ID_FILE_OPEN — пункт Open меню File. Значение идентификатора ресурса можно просмотреть и при желании отредактировать в поле ID диалогового окна Properties.
- *Поле* Caption *(Надпись)*. Это текст, который определяет изображение пункта меню. На рис. B.6 *надпись* видна в поле Caption диалогового окна Properties — это &Open...\tCtrl+0. Символ & означает, что символ 0 будет подчеркнут в изображении меню. Когда меню будет выведено на экран в работающем приложении, этот пункт можно будет выбрать, нажав <O>. Символ \t — табуляция, а Ctrl+0 — акселератор для этого пункта меню (см. рис. B.3).

- *Поле* Prompt *(Пояснение)*. Пояснение появляется в строке состояния, когда выделен пункт меню или курсор находится над ассоциированной (соответствующей) пиктограммой панели инструментов. На рис. В.6 текст *пояснения* виден в поле Prompt диалогового окна Properties — это `Open an existing document\nOpen` (Открытие существующего документа\nОткрыть). В строке состояния отобразится только часть пояснения, предшествующая символу новой строки (`\n`). Вторая часть текста пояснения — `Open` — будет выведена в контекстном окне указателя, которое появится в работающем приложении если пользователь установит указатель мыши на соответствующей пиктограмме панели инструментов, т.е. на пиктограмме, связанной с тем же идентификатором ресурса, что и выделенный пункт меню. Все эти функциональные возможности обеспечиваются автоматически интегрированной системой Visual C++ и MFC.

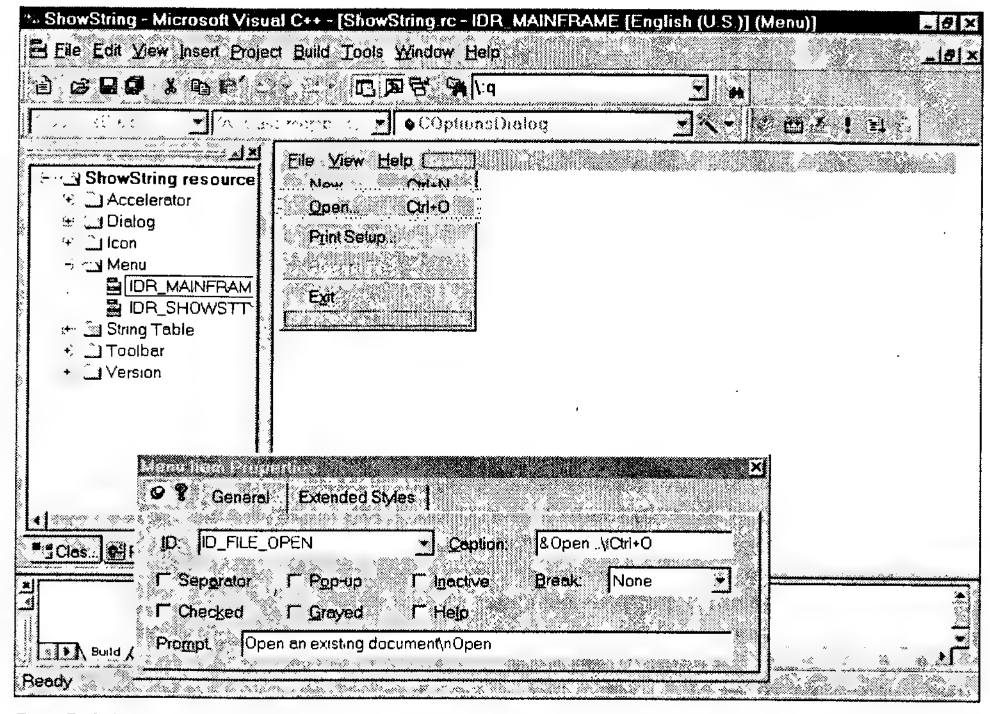


Рис. В.6. Меню — средство передачи приложению команд пользователя

Таблица строк

Таблица строк — это список строковых констант в разрабатываемом приложении. Ко многим строкам, таким как статические текстовые поля в диалоговом окне или пояснения для пунктов меню, доступ может быть осуществлен значительно проще, чем через таблицу строк, но к некоторым из строк можно добраться только через нее. Например, имя по умолчанию или значение может храниться в таблице строк и изменяться без перекомпилирования всей программы (хотя ресурсы должны быть перекомпилированы, а проект скомпонован повторно). Конечно, любую строковую константу можно просто включить в текст программы, но затем замена этих констант потребует полной повторной трансляции.

На рис. В.7 приведена таблица строк для нашего приложения. Для изменения таблицы строк необходимо вызвать диалоговое окно Property и отредактировать текст в поле Caption. Строка не может быть изменена прямо в окне рабочей области.

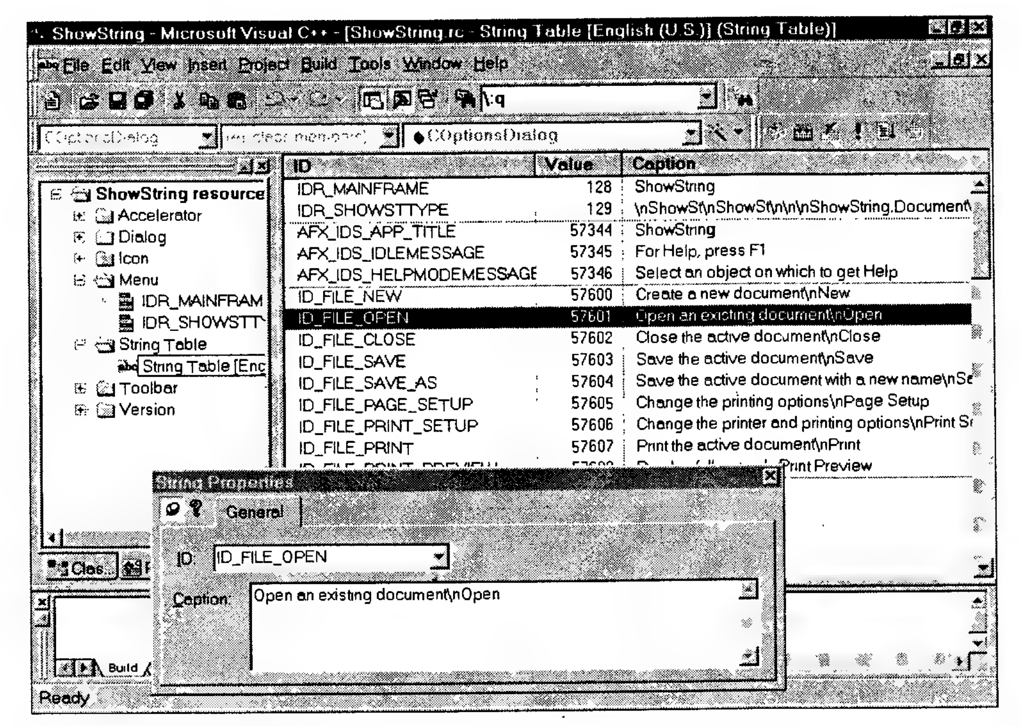


Рис. В.7. В таблице строк хранятся все текстовые константы приложения

Панели инструментов

Панели инструментов — это набор пиктограмм, который, как правило, размещается ниже строки меню приложения. Каждая пиктограмма связана с пунктом меню, и ее "внешний вид" зависит от состояния пункта меню. Если пункт меню уже выбран, соответствующая пиктограмма обычно имеет вид *вдавленной* кнопки. Таким образом, пиктограммы служат как индикаторами, так и средствами формирования команд приложению.

Пиктограммы панели инструментов состоят из двух компонентов — растровой картинки (собственно пиктограммы) и идентификатора ресурса. Когда пользователь щелкнет на пиктограмме, произойдет то же самое, что и при выборе пункта меню с тем же идентификатором ресурса. На рис. В.8 приведена типовая (стандартная) панель инструментов и свойства ее пиктограммы File⇨Open. Сейчас можно поменять идентификатор ресурса любой пиктограммы или отредактировать ее изображение теми же средствами, которые применяются для редактирования обычных пиктограмм.

Информация о версии

В хороших программах для инсталляции разработанного приложения на компьютере пользователя применяется ресурс *информации о версии*. Например, если пользователь инсталлирует приложение, которое ранее уже было установлено, программа инсталляции может некоторые файлы не копировать и предупредить пользователя, если он попытается установить старую версию поверх новой, и т.д.

При создании приложения с помощью `AppWizard` информация о версии, подобная приведенной на рис. В.9, формируется автоматически. Перед тем как изменить ее, разберитесь, как же эту информацию использует программа установки.

Рис. В.8. Пиктограммы панели инструментов связаны с пунктами меню через идентификатор ресурса

Рис. В.9. Информация о версии используется программой инсталляции

Просмотр текста программы, организованный соответственно классам

При выборе вкладки ClassView у разработчика появляется возможность просмотреть в окне Workspace структуру классов, использованных в приложении. Для каждого класса в дереве списка показаны члены — переменные и методы (рис. В.10). Методы показаны первыми в списке и рядом с ними стоит розовая пиктограмма. Далее в списке следуют члены-переменные, отмеченные бирюзовой пиктограммой. Защищенные (protected) члены помечены ключом рядом с пиктограммой, закрытые (private) — замком.

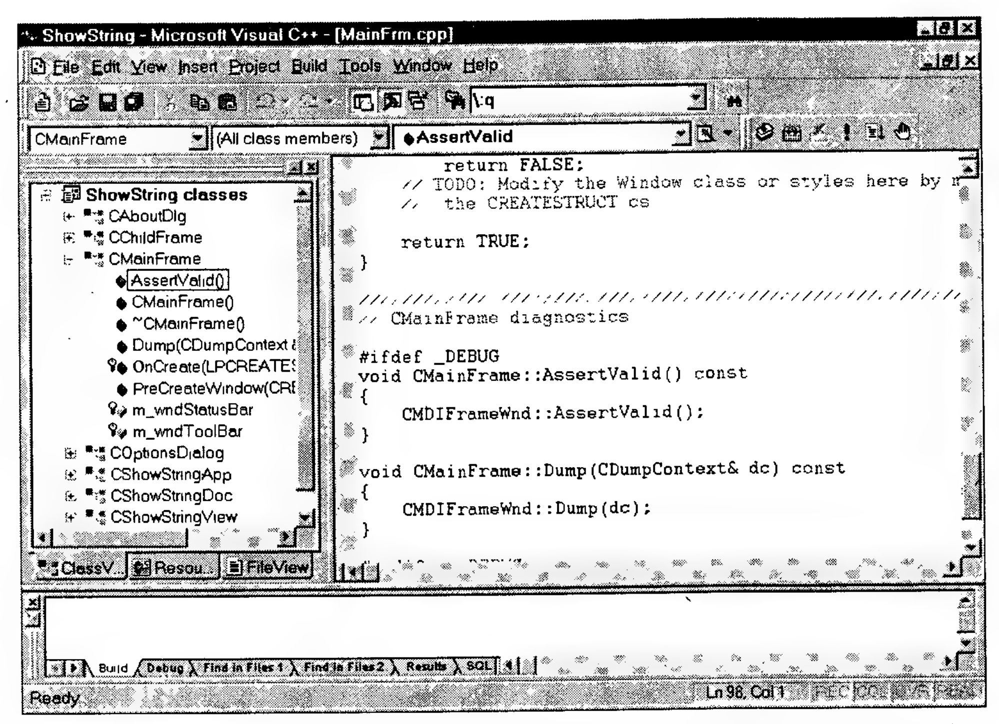


Рис. В.10. На вкладке ClassView можно просмотреть методы и переменные в каждом классе приложения

Двойной щелчок на имени метода позволяет вывести в окне редактора кода текст соответствующей функции. Двойной щелчок на идентификаторе переменной приводит к выводу в окне редактора кода текста файла заголовка, в котором объявлена переменная.

Если щелкнуть на имени класса правой кнопкой мыши, появится контекстное меню, показанное на рис. В.11. В меню имеются следующие пункты.

- Go to Definition (Переход к объявлению). Открывает файл заголовка (.h) в том месте, где объявлен этот класс.
- Go to Dialog Editor (Переход к редактору диалоговых окон). Для классов диалоговых окон открывает диалоговое окно редактирования ресурсов.
- Add Member Function (Добавление члена-функции). Открывает диалоговое окно Add Member Function, приведенное на рис. В.12.

Рис. B.11. В контекстном меню ClassView находятся команды, чаще всего используемые при работе с классами

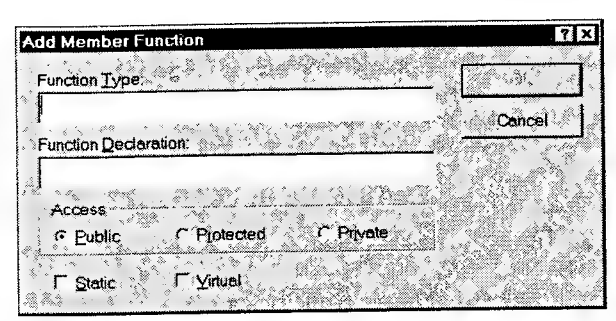


Рис. B.12. Теперь при использовании окна Add Member Function вы никогда не забудете о составляющих объявления функции

- Add Member Variable (Добавление члена-переменной). Открывает диалоговое окно Add Member Variable, приведенное на рис. B.13. Добавляет объявление переменной в файл заголовка.
- Add Virtual Function (Добавление виртуальной функции). Открывает диалоговое окно Add Virtual Function, которое подробно рассматривается в главе 3.
- Add Windows Message Handler (Добавление обработчика сообщения Windows). Открывает диалоговое окно New Windows Message and Event Handlers, которое подробно рассматривается в главе 3.

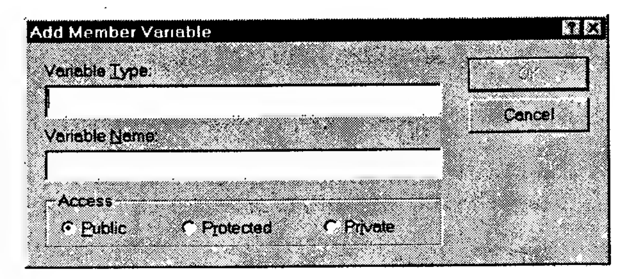


Рис. В.13. Упрощенная процедура вставки члена-переменной

- References (Ссылки). Выдает список ссылок на те строки в тексте программы, в которых упоминается имя класса в приложении. Обычно имя класса встречается в объявлениях экземпляров класса, но эта команда находит также места, в которых имя класса передается в качестве параметра функции или макросу.
- Derived Classes (Производные классы). Выдает список всех членов — функций и переменных — класса и список других классов, использующих данный класс как базовый, а также связанную с этим информацию.
- Base Classes (Базовые классы). Выводит список всех членов — функций и переменных — класса и список базовых классов для данного класса, а также связанную с этим информацию.
- Add to Gallery (Включение в `Component Gallery`). Добавляет этот класс в библиотеку `Component Gallery`, которая рассматривается в главе 25.
- New Folder (Новая папка). Создает папку, в которую можно поместить классы. Это помогает организовать проект с большим количеством классов.
- Group by Access (Группировка по доступу). Переупорядочивает список. По умолчанию методы класса располагаются в алфавитном порядке, за ними следует список членов-переменных, также расположенных в алфавитном порядке. Если же включить эту опцию, то первыми будут следовать методы, упорядоченные по типу доступа (открытые, затем защищенные, затем закрытые, причем в каждом разделе они упорядочены по алфавиту), а затем — переменные (снова открытые, затем защищенные, затем закрытые члены-переменные, причем в каждом разделе в алфавитном порядке).
- Docking View (Стационарная компоновка окон). Переключает режим стационарной/плавающей компоновки окна Workspace в пределах экрана среды разработки.
- Hide (Спрятать). Прячет окно Workspace. Чтобы снова его открыть, выберите в меню Visual Studio команду View⇨Workspace.
- Properties (Свойства). Отображает свойства класса (имя, базовый класс).



Возле тех пунктов меню, которые дублированы на панели инструментов, изображены соответствующие пиктограммы. Постарайтесь их запомнить. Возможно, когда в следующий раз вы захотите выбрать эту команду, быстрее будет щелкнуть на соответствующей пиктограмме.

Щелчок правой кнопкой мыши на имени *метода* класса приведет к появлению еще одного контекстного меню со следующими пунктами.

- Go To Definition (Переход к определению). Открывает файл реализации класса (`.cpp`) в том месте, где реализован этот метод.
- Go To Declaration (Переход к объявлению). Открывает файл заголовка (`.h`) в том месте, где объявлен этот метод.
- Delete (Удаление). Удаляет метод из списка членов класса.

- Set Breakpoint (Установка точек останова). Устанавливает точку останова. Назначение и работа с точками останова обсуждается в приложении Г.
- References (Ссылки). Выдает список ссылок на те строки в тексте программы, в которых есть обращение к этой функции.
- Calls (Вызовы). Выводит на экран иерархический список всех функций, которые этот метод вызывает. Как всякий иерархический список, его можно сворачивать и разворачивать. На рис. В.14 показан пример окна Call Graph.

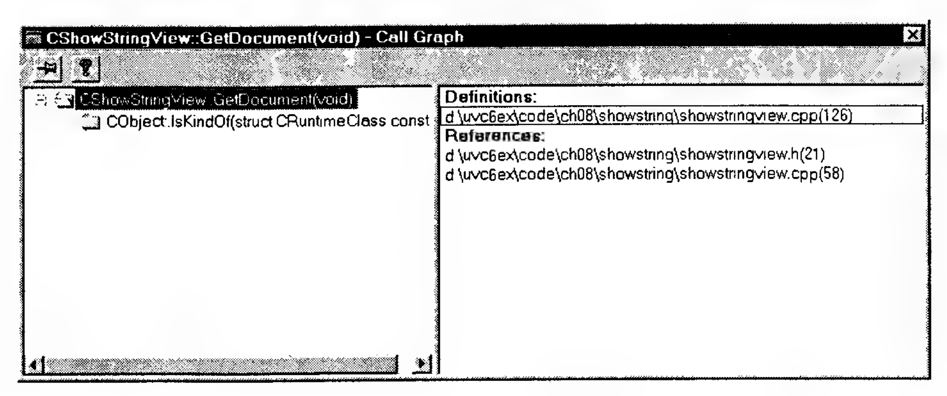


Рис. В.14. В окне Call Graph содержится список всех функций, которые этот метод вызывает

- Called By (Кем вызывается). Выводит на экран список всех функций, которые вызывают этот метод.
- New Folder (Новая папка). Создает папку, в которую можно поместить классы. Это помогает организовать проект с большим количеством классов.
- Group by Access (Группировка по доступу). Переупорядочивает список. По умолчанию методы класса располагаются в алфавитном порядке, за ними следует список членов-переменных, также расположенных в алфавитном порядке. Если же включить эту опцию, то первыми будут следовать методы, упорядоченные по типу доступа (открытые, затем защищенные, затем закрытые, причем в каждом разделе они упорядочены по алфавиту), а затем — переменные (снова открытые, затем защищенные, затем закрытые члены-переменные, причем в каждом разделе в алфавитном порядке).
- Docking View (Стационарная компоновка окон). Переключает режим стационарной/плавающей компоновки окна Workspace в пределах экрана среды разработки.
- Hide (Спрятать). Прячет окно Workspace. Чтобы снова его открыть, выберите View⇨Workspace.
- Properties (Свойства). Отображает свойства класса (имя, базовый класс).

Щелчок правой кнопкой мыши на имени члена-переменной приведет к появлению еще одного контекстного меню со следующими пунктами.

- Go To Definition (Переход к объявлению). Открывает файл заголовка (.h) в том месте, где объявлена эта переменная.
- References (Ссылки). Выдает список ссылок на те строки в тексте программы, в которых есть обращение к этой переменной.
- New Folder (Новая папка). Создает папку, в которую можно поместить классы. Это помогает организовать проект с большим количеством классов.
- Group by Access (Группировка по доступу). Переупорядочивает список. По умолчанию методы класса располагаются в алфавитном порядке, за ними следует список членов-переменных, также расположенных в алфавитном порядке. Если же включить эту

опцию, то первыми будут следовать методы, упорядоченные по типу доступа (открытые, затем защищенные, затем закрытые, причем в каждом разделе они упорядочены по алфавиту), а затем — переменные (снова открытые, затем защищенные, затем закрытые члены-переменные, причем в каждом разделе в алфавитном порядке).

- Docking View (Стационарная компоновка окон). Переключает режим стационарной/плавающей компоновки окна Workspace в пределах экрана среды разработки.
- Hide (Спрятать). Прячет окно Workspace. Чтобы снова его открыть, выберите View⇨Workspace.
- Properties (Свойства). Отображает свойства класса (имя, базовый класс).

Когда в основной рабочей области выведен текст программы или файла заголовка, его можно редактировать, как описано ниже, в разделе *Редактирование текстов программ.*

Просмотр файлов программ

Организация информации на вкладке FileView во многом сходна с ClassView в том смысле, что можно просматривать и редактировать тексты программ и файлы заголовков (рис. B.15). Однако FileView предоставляет доступ к тем фрагментам программы, которые расположены вне определений класса, а также облегчает открытие текстовых файлов, не являющихся собственно программами (файлов ресурсов и обычного текста).

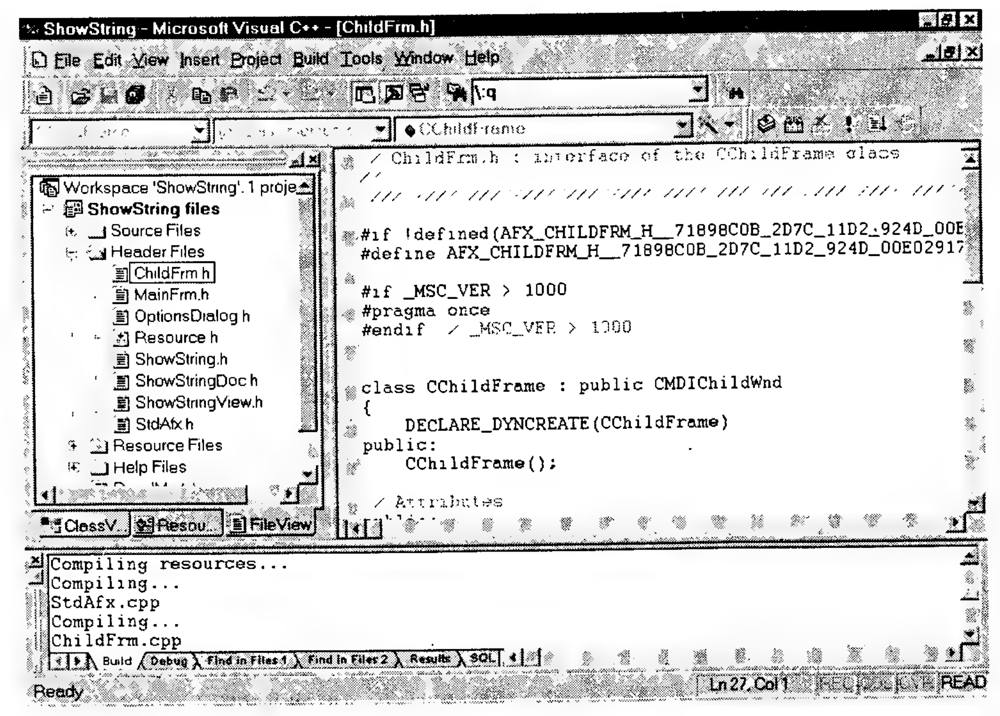


Рис. B.15. FileView позволяет просматривать тексты программ и файлы заголовков

В окне Workspace представлено дерево файлов проекта. По умолчанию файлы разделяются на следующие категории: собственно тексты программ (Source Files), файлы заголовков (Header Files), файлы ресурсов (Resource Files), файлы справки (Help Files) и внешние зависимости (External Dependences). Категория файлов справки включается в список в том

случае, если проект снабжен справкой. Можно добавлять собственные категории, щелкнув правой кнопкой мыши в любом месте окна FileView и выбрав из контекстного меню New Folder. После этого определите, файлы с каким расширением войдут в новую категорию.

Двойной щелчок на имени файла позволяет вывести этот файл в основной рабочей области. Затем можно редактировать файл (даже если он не является текстом на языке C++), как описано ниже, в разделе *Редактирование текстов программ*.

Выходные сообщения и сообщения об ошибках

В самом низу экрана Visual Studio находится окно вывода Output. В нем выводятся различные сообщения компонентов Visual Studio, в том числе сообщения об ошибках.



Если на экране отсутствует окно Output, для его восстановления выберите команду View⇨Output.

В окне Output имеется пять вкладок.

- Build (Компиляция и компоновка). Отображает результаты компиляции и компоновки.
- Debug (Отладка). Применяется при отладке, как описано в приложении Г.
- Find in Files 1 (Поиск в файлах 1). Отображает результаты поиска во время выполнения команды Find in Files; обсуждается ниже в этой главе.
- Find in Files 2 (Поиск в файлах 2). Альтернативный вариант окна для результатов выполнения команды Find in Files. В этом варианте можно сохранить результаты предыдущего поиска.
- Results (Результаты). Отображает результаты работы таких инструментальных средств, как система отображения профиля программы, которая рассматривается в главе 24.

Если вы инсталлировали Enterprize Edition Visual C++, то в вашем Visual Studio имеется шестая вкладка SQL Debugging (Отладка SQL). Дополнительная информация содержится в главе 23.

Редактирование текстов программ

Для большинства программистов редактирование программ является основной операцией в процессе разработки программы в рамках интегрированной среды. Если вы уже пользовались каким-либо редактором или текстовым процессором, то довольно скоро освоите и редактор Visual Studio. Вам необходимо уметь вводить текст программы, фиксировать ошибки, перемещаться по файлам текста программы и заголовка, используя те средств Windows, которые вам нужны. Но так как это редактор, ориентированный на тексты программ, в нем есть несколько интересных функций, о которых необходимо иметь представление.

Базовые операции ввода и редактирования текста

Чтобы добавить текст в файл, щелкните кнопкой мыши в том месте, куда вы хотите добавить текст, и начните ввод. По умолчанию редактор находится в *режиме вставки*, который означает, что новый текст "расталкивает" старый. Для включения *режима замены* (Overstrike

mode) необходимо нажать клавишу <Ins>. Теперь новый текст вводится поверх старого. Индикатор OVR в строке состояния напоминает о том, что вы находитесь в режиме замены. Если снова нажать клавишу <Ins>, то можно вернуться в режим вставки. Перемещаться по тексту файла можно либо с помощью мыши, либо с помощью клавиш управления курсором. Для перехода на новую страницу используются клавиши <PgUp> и <PgDn> или полоса прокрутки, расположенная справа от рабочей зоны окна.

По умолчанию окно для редактируемого файла раскрывается на всю рабочую зону Visual Studio. Чтобы показать файл в меньшем окне, щелкните на кнопке Restore справа вверху, непосредственно под строкой заголовка Visual Studio. Если имеется сразу несколько открытых файлов, можете скомпоновать их на экране в виде набора перекрывающихся окон, как показано на рис. В.16.

Рис. В.16. Файлы выведены в MDI-окнах так, что можно работать сразу с несколькими файлами

Работа с блоками текста

Часто возникает необходимость выполнить действия с целым блоком текста. Сначала вы выбираете текст, щелкнув мышью на одном конце блока, и, удерживая кнопку мыши нажатой, передвигаете ее указатель на другой конец блока и там отпускаете кнопку мыши. Эта операция знакома всем, кто когда-либо имел дело с приложениями Windows. На этом этапе вы можете скопировать блок в системный буфер `Clipboard`, заменить его введенным текстом или текстом из `Clipboard`. Можно и удалить выделенный блок.



Чтобы выбрать столбцы текста, как показано на рис. В.17, необходимо при выборе удерживать нажатой клавишу <Alt>.

Рис. В.17. Выбор столбцов делает фиксацию отступов вправо намного проще. Выбор блока осуществляется при нажатой клавише <Alt>

Синтаксическая раскраска

Скорее всего, вы уже обратили внимание на раскраску, применяемую при отображении текста программы. Visual Studio выделяет элементы программы с помощью *синтаксической раскраски*. По умолчанию текст программы окрашен в ***черный*** цвет с комментариями ***зеленого*** цвета и ключевыми словами (служебными словами в C++ наподобие `public`, `private`, `new` и `int`) ***голубого*** цвета. Можно также сформировать специальные расцветки для строковых переменных, чисел, терминальных операторов (вида + или -), используя вкладку Format или диалоговое окно Options. Для этого необходимо выбрать команду Tools⇨Options.

Синтаксическая раскраска помогает замечать ошибки, допущенные по небрежности. Если вы забыли закрыть комментарий операторной скобкой, как того требует прежний стиль С-программ, большая зеленая полоса в файле сразу укажет на ошибку, которую необходимо исправить. Если вместо типа `int` вы написали `inr`, *inr* будет выведено не голубым цветом, что явится для вас предупреждением о несоответствии ключевого слова. Из всего этого следует, что такое средство, как синтаксическая раскраска, позволяет избежать многих ошибок еще до компиляции программы.



Если для формирования Web-страниц вы до сих пор изредка используете `Notepad`, в котором можете видеть теги, вам предстоит приятно удивиться. Откройте файл HTML в Visual Studio и посмотрите на синтаксическую раскраску в действии. После этого вы навсегда отправите `Notepad` на заслуженный отдых.

Контекстное меню

Многие часто встречающиеся операции можно выполнять, обратившись к контекстному меню, которое появляется, если щелкнуть правой кнопкой мыши на поле редактируемого файла. Это меню состоит из следующих пунктов.

- Cut. Удаляет выбранный текст в системный буфер Clipboard.
- Copy. Копирует выбранный текст в системный буфер Clipboard.
- Paste. Заменяет выбранный текст содержимым системного буфера Clipboard или, если текст не выбран, вставляет содержимое системного буфера Clipboard в то место, где находится курсор.
- Insert File Into Project. Включает редактируемый файл в состав открытого проекта.
- Check Out. При использовании Visual Source Safe эта команда позволяет маркировать файл, как измененный вами в текущем сеансе.
- Open. Открывает файл, на имя которого указывает курсор. Особенно удобно обращаться к этой операции для вызова на экран файлов заголовков, потому что не нужно вспоминать, в каких папках они находятся.
- List members. Выводит список всех членов объекта (переменных и методов), на имя которого указывает курсор.
- Type Info. Выводит контекстное окно с типом переменной или функции (типом результата, возвращаемого функцией).
- Parameter Info. Выводит контекстное окно с типами аргументов функции.
- Complete World. Активизирует функцию AutoComplete, которая помогает завершить ввод имен функций или переменных, частично введенных с клавиатуры.
- Go To Definition. Открывает файл, в котором определен элемент, на который указывает курсор (файл заголовка — для переменной, файл текста программы — для функции).
- Go To Reference. Перемещает курсор на следующую ссылку на переменную или функцию, имя которой расположено под курсором.
- Insert/Remove Breakpoint. Помещает точку останова на оператор, на который указывает курсор, или удаляет ее, если она уже установлена.
- Enable Breakpoint. Разблокирует заблокированную точку останова (работа с точками останова рассматривается в приложении Г).
- ClassWizard. Запускает ClassWizard.
- Properties. Выводит на экран список свойств.

Не все пункты включаются сразу; например, Cut и Copy включаются только тогда, когда имеется выбранный фрагмент текста. Команда Insert File Into Project разрешена только в случае, если редактируемый файл не включен в открытый проект. Все эти команды доступны как с помощью основного меню, так и через пиктограммы панели инструментов. Более подробно они будут рассмотрены ниже в этой главе.

Система меню

Visual Studio имеет много меню. Некоторые команды находятся на третьем или четвертом уровне иерархической структуры меню. В большинстве случаев ту же задачу можно решить значительно быстрее, но начинающему пользователю меню легче изучить, поскольку есть

возможность читать наименования пунктов меню вместо того, чтобы запоминать комбинации клавиш ускоренного доступа. В строке меню Visual Studio есть девять меню.

- File — меню операций, связанных с целыми файлами (открытие, закрытие и печать).
- Edit — меню операций копирования, удаления, предыстории, поиска и перемещения по файлу.
- View — меню операций управления видами и средствами Visual Studio, включая панели инструментов и подокна типа окна Workspace.
- Insert — меню команд включения файлов или их компонентов в разрабатываемый проект.
- Project — меню команд управления разрабатываемым проектом в целом.
- Build — меню команд компиляции, компоновки и отладки.
- Tools — меню команд настройки Visual Studio и доступа к автономным утилитам.
- Window — меню выбора окон, когда они распахнуты на все рабочее поле или свернуты в пиктограмму.
- Help — меню команд обращения в системе `InfoViewer` (сюда не входят операции обращения к обычной оперативной справке).

Ниже мы рассмотрим каждое из меню. При этом будут упомянуты и комбинации клавиш ускоренного вызова, и пиктограммы панели инструментов, дублирующие соответствующие команды (если таковые существуют в Visual Studio).

Меню File

Меню File, приведенное на рис. В.18, содержит большинство команд, имеющих отношение к файлам или проектам целиком.

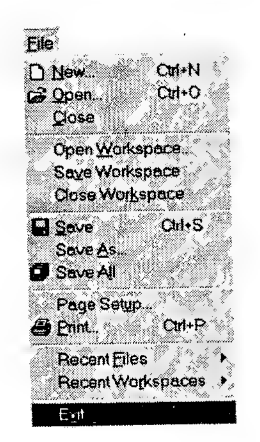


Рис. В.18. Меню File включает операции с файлами Open, Close и Print

File⇨New (<Ctrl+N>)

Выбор этого пункта меню приводит к появлению диалогового окна New, показанного на рис. В.19. Это окно используется для создания новых файлов, проектов, описаний рабочей среды и других документов. Вкладка Project применяется для запуска `AppWizard`, о чем впервые речь шла в главе 1.

Это диалоговое окно позволяет сразу создать пустой файл, присвоить ему имя и включить в разрабатываемый проект.

File⇨Open (<Ctrl+O>)

Выбор этого пункта меню приводит к появлению диалогового окна Open, показанного на рис. В.20. (Это стандартное диалоговое окно Windows File Open, поэтому оно хорошо знакомо всем, кто когда-либо садился за компьютер.) По умолчанию искомый файл относится к типу `Common Files` (файлы общего назначения) с расширениями `.c`, `.cpp`,`.h` , `.cxx`, `.rc`, `.tli`, `.tlh`, `.inl`. Если щелкнуть кнопкой мыши на выпадающем списке, можно открыть почти любой файл, включая и выполняемые.

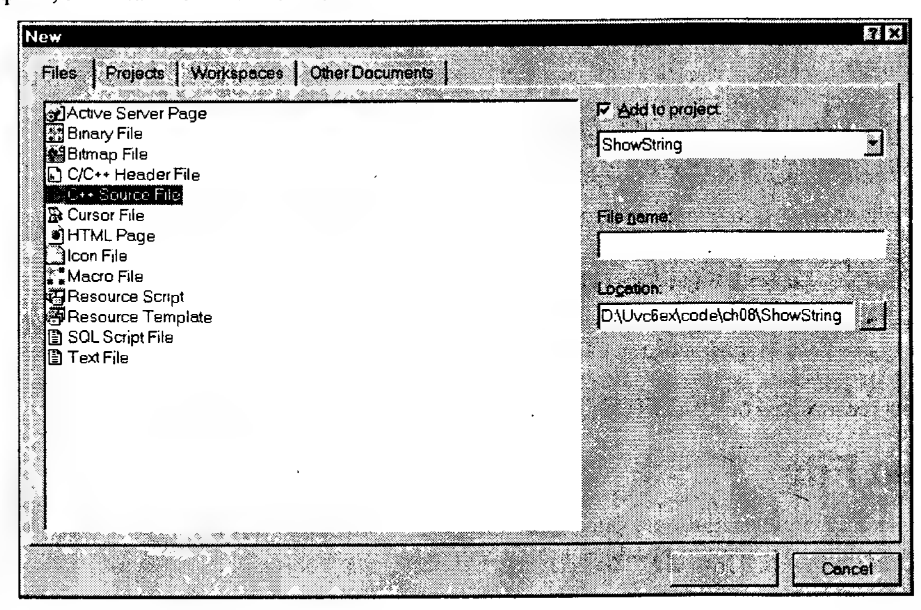


Рис. В.19. Диалоговое окно New *используется для создания новых файлов или рабочей среды*

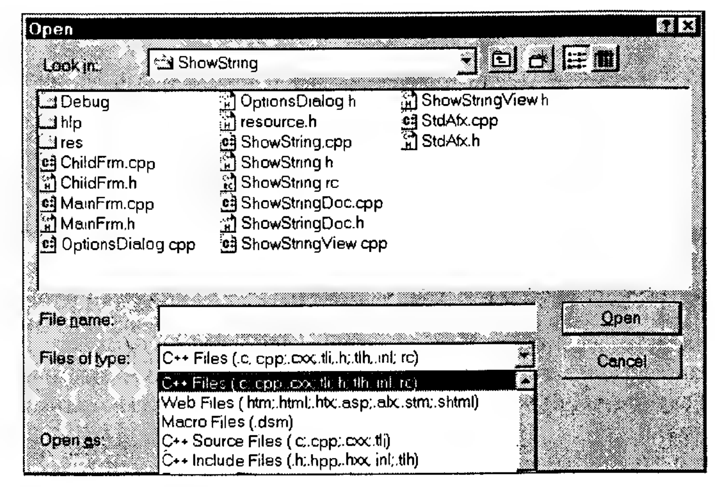


Рис. В.20. Знакомое диалоговое окно File Open *используется для того, чтобы открыть файл любого типа*



Не забудьте о списке недавно открытых файлов в меню File. Он поможет вам избежать утомительной работы с клавиатурой или мышью.

File⇨Close

Команда File⇨Close позволяет закрыть файл, который имеет фокус. Если таких файлов нет, соответствующая надпись в меню будет затенена, а команда заблокирована. Файл можно закрыть, щелкнув на кнопке Cancel, обозначенной символом X, в правом верхнем углу окна. Файл также можно закрыть, сделав двойной щелчок на пиктограмме в левом верхнем углу окна.

File⇨Open Workspace

Эта команда используется для того, чтобы открыть описание рабочей среды. Того же можно достичь, выбрав File⇨Open и тип файла `Project Workspace`, но быстрее будет воспользоваться File⇨Open Workspace.

File⇨Save Workspace

Эта команда используется для того, чтобы сохранить описание рабочей среды и все открытые в текущий момент файлы.

File⇨Close Workspace

Эта команда используется для того, чтобы закрыть описание рабочей среды. Описание текущей рабочей среды закрывается автоматически, когда создается новый проект или открывается новое описание рабочей среды, так что этот пункт меню используется довольно редко.

File⇨Save (<Ctrl+S>)

Команда используется для записи файла, имеющего в данный момент фокус; если фокус отсутствует, соответствующая надпись в меню будет затенена, а команда заблокирована. На стандартной панели инструментов также имеется пиктограмма Save.

File⇨Save As

Эта команда используется для записи файла под новым именем. По этой команде будет записан файл, имеющий в данный момент фокус; если фокус отсутствует, соответствующая надпись в меню будет затенена, а команда заблокирована.

File⇨Save All

Эта команда позволяет сохранить все открытые файлы. Все файлы сохраняются непосредственно перед компиляцией и когда закрывается приложение, но, если компиляция выполняется не часто и есть довольно много изменений в файлах, рекомендуется их сохранять каждые 10–15 минут. (Если вас не волнует возможная, например, из-за сбоя питания потеря результатов выполненной работы, можете делать это реже.)

File⇨Page Setup

По этой команде на экран выводится диалоговое окно Page Setup, показанное на рис. В.21. Здесь определяются верхний (Header) и нижний (Footer) колонтитулы, а также размеры пустых полей — слева (Left), справа (Right), сверху (Top) и снизу (Bottom). Верхний и нижний колонтитулы могут содержать любой текст, включая одно или более специальных

полей, которые добавляются, если щелкнуть на стрелке рядом с окном редактирования или ввести коды самому. Ниже перечислены коды специальных полей.

- Имя распечатываемого файла (&f)
- Номер текущей страницы (&p)
- Время печати страницы (&t)
- Дата печати страницы (&d)
- Выравнивание левой позиции строки (&l)
- Выравнивание правой позиции строки (&r)
- Центрирование строки (этот вариант выравнивания устанавливается по умолчанию) (&c)

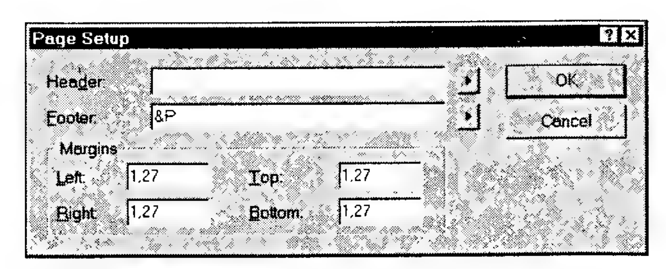


Рис. В.21. В диалоговом окне Page Setup *устанавливаются параметры печати страницы по вашему усмотрению*

File⇨Print (<Ctrl+P>)

При выборе этого пункта печатается файл, на имени которого находится фокус. Параметры печати установлены согласно Page Setup. Если в фокусе нет файла, соответствующая надпись в меню будет затенена, а команда заблокирована. В диалоговом окне Print, показанном на рис. В.22, вы должны подтвердить тип принтера, выбранного для печати. Если есть выделенный текст, переключатель Selection (Отмеченный) разблокируется. Эта опция позволяет печатать только выделенный текст. В противном случае включается лишь переключатель All (Все) и печатается весь файл. Если перед выбором File⇨Print вы забыли установить колонтитулы и поля, кнопка Setup приведет вас в диалоговое окно Page Setup. Возможность печати отдельных страниц или остановки печати после того, как она началась, отсутствует.

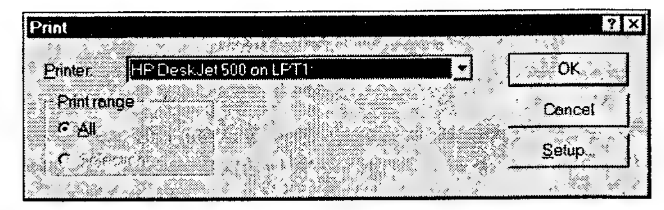


Рис. В.22. В диалоговом окне Print *нужно подтвердить свое согласие печатать файл*

Recent Files и Recent Workspaces

Пункты Recent Files (Свежие файлы) и Recent Workspaces (Свежие рабочие среды), которые находятся в меню между пунктами Print и Exit, представляют собой входы в следующий уровень меню. Пункты во вторичных меню содержат имена файлов и описаний рабочей среды, которые были открыты последними. При работе над несколькими проектами

параллельно использование этих пунктов дает ощутимую экономию времени. А потому, прежде чем щелкнуть на File⇨Open, подумайте, не может ли этот файл находиться в списке Recent Files или Recent Workspaces. Меню не всегда выполняют роль "указателя" пути.

File⇨Exit

Чаще всего используемая команда меню Windows позволяет закрыть Visual Studio. Можно также щелкнуть на кнопке закрытия в правом верхнем углу или сделать двойной щелчок на пиктограмме, которая представляет системное меню, слева в строке заголовка. Если вы сделали изменения и не сохранили результат, у вас еще будет возможность сохранить каждый измененный файл перед закрытием Visual Studio.

Меню Edit

В меню Edit (Правка), приведенном на рис. В.23, собраны команды, связанные с редактированием текста в файле.

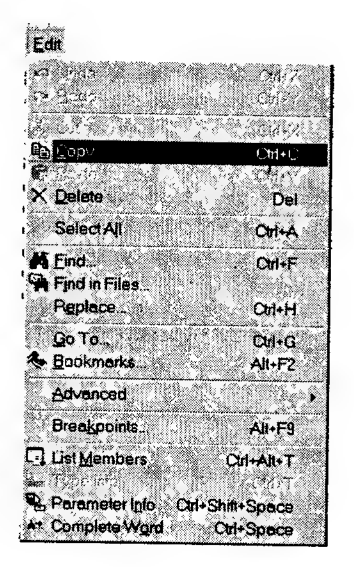


Рис. В.23. В меню Edit содержатся пункты, позволяющие изменить текст файла

Edit⇨Undo (<Ctrl+Z>)

Команда Undo (Откат) отменяет операции, выполненные ранее. Большинство операций, таких как редактирование и удаление текста, может отменяться. Когда Undo заблокирована, значит, нет ничего, что надо отменять, или что вы не можете отменить последнюю операцию.

На стандартной панели инструментов также имеется пиктограмма Undo. Щелчок на пиктограмме приводит к отображению стека (списка в обратном порядке от самых свежих до самых поздних) операций, которые могут быть отменены. Необходимо выделить непрерывный интервал пунктов от первого, второго и т.д. Нельзя пропускать элементы, а потом снова их выбирать.

Edit⇨Redo (<Ctrl+Y>)

Как только вы отменили некоторую операцию, ее имя переходит из Undo в список Redo (Восстановить). (Пиктограмма Redo — следующая за Undo на панели инструментов.) Если вы в спешке удалили немного больше, чем требовалось, выберите Edit⇨Redo — и все вернется на круги своя.

Edit⇨Cut (<Ctrl+X>)

Эта команда позволяет переносить текущий выделенный текст в системный буфер Clipboard, а сам текст удалить из файла. Пиктограмма Cut (она имеет вид ножниц) находится на стандартной панели инструментов.

Edit⇨Copy (<Ctrl+C>)

Пиктограммы редактирования сгруппированы после "ножниц". Edit⇨Copy копирует только что выбранный текст или элемент в системный буфер Windows Clipboard.

Edit⇨Paste (<Ctrl+V>)

По этой команде содержимое Clipboard копируется в то место, где находится курсор, или заменяет выделенный текст. Пункт и пиктограмма Paste (Вставка) заблокированы, если в Clipboard нет ничего подходящего по формату для вставки в тот файл, на котором стоит фокус. В дополнение к тексту можно копировать элементы меню, диалоговых окон и другие ресурсы. Пиктограмма Paste находится на стандартной панели инструментов.

Edit⇨Delete (<Del>)

Команда Edit⇨Delete позволяет удалить отмеченный текст или элемент. Если то, что вы удалили, восстановимо, последняя операция добавляется в стек пиктограммы Undo. Удаленный текст не поступает в буфер Clipboard и может быть восстановлен только путем отмены удаления.

Edit⇨Select All (<Ctrl+A>)

Эта команда позволяет выделить все содержимое файла, который находится в фокусе. Например, если в фокусе находится файл текста программы, выделяется весь текст, если диалоговое окно, выделяются все элементы управления.

Для выделения множества элементов в диалоговом окне можно щелкнуть на первом элементе и затем нажать <Ctrl> и щелкать на каждом из оставшихся элементов. Часто для быстрого выделения лучше всего использовать Edit⇨Select All, а затем нажать <Ctrl> и щелкать на каждом элементе, который нужно исключить из выделенной области.

Edit⇨Find (<Ctrl+F>)

Диалоговое окно Find (Поиск), приведенное на рис. В.24, разрешает поиск текста внутри файла, который имеет фокус. Введите в поле Find what (Что искать) слово или фразу. Следующие флажки используются для установки опций поиска.

- Match whole word only (Только слово целиком). Если эта опция установлена, то при table в поле Find what в тексте будет найдено только table, но не suitable или tables.
- Match case (Учитывать регистр). Если эта опция установлена, то при Chapter в поле Find what в тексте будет найдено только Chapter, но не CHAPTER или chapter, т.е. регистр букв в образце должен совпадать с регистром в найденном тексте.
- Regular expression (Регулярное выражение). Содержимое поля Find what считается *регулярным выражением*, если выбрана эта опция.
- Search all open documents (Искать во всех открытых документах). Опция включает операцию поиска во все документы, которые открыты на данный момент.
- Direction (Направление). Для поиска в обратном порядке (назад) выберите переключатель Up (Вверх), а переключатель Down (Вниз) — для поиска вперед по файлу.

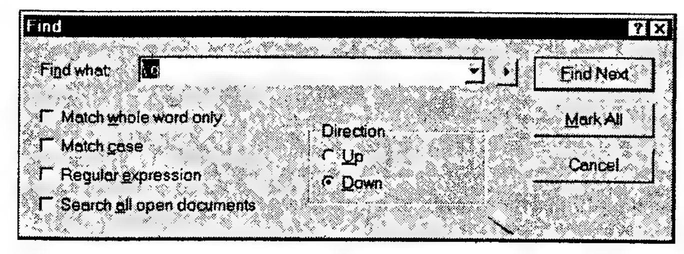


Рис. В.24. Диалоговое окно Find *используется для поиска текста в файле, на имени которого находится фокус*

Обычная методика работы с диалоговым окном Find — ввести некоторый текст и щелкать на кнопке Find Next (Найти следующий), пока не найдется именно тот экземпляр образца, который вы ищете. Но можно объединить функцию поиска и расстановку закладок (такой вариант рассматривается в этом разделе немного ниже) и поместить закладку в каждую строку текста, в которой встречается образец. Чтобы добавить временные, неименованные закладки на строки с найденным образцом, щелкните на кнопке Mark All (Маркировать все) в диалоговом окне Find; эти строки будут отмечены голубым овалом по краю.

На стандартной панели инструментов имеется текстовое поле Find. Введите в это окно образец и нажмите клавишу <Enter> для поиска вперед. Можно использовать и регулярные выражения, если вы соответственно настроили эту опцию в диалоговом окне Find. Для повторного поиска щелкните на пиктограмме Find Next (Найти следующий) или Find Previous (Найти предыдущий) стандартной панели инструментов (они представляют собой пиктограммы с биноклями со стрелкой в направлении часовой стрелки и против часовой стрелки соответственно).

Регулярные выражения

Можно расширить возможности операции поиска и замены в Visual Studio с помощью регулярных выражений. Например, если требуется найти образец только в конце строки текста или строки, незначительно отличающейся от образца, это можно сделать, сконструировав подходящее регулярное выражение. Затем его нужно ввести в диалоговое окно Find и дать инструкцию Visual Studio использовать для поиска регулярное выражение. Регулярное выражение представляет собой некоторый текст, включающий специальные символы. Эти символы представляют элементы текста, которые не могут быть введены с клавиатуры, например *конец строки*, *любое число* и *три заглавные буквы*.

При использовании регулярных выражений некоторые символы меняют свое обычное значение и превращаются в один или несколько других символов. Регулярные выражения в Visual Studio построены из обычных символов, скомбинированных со специальными элементами, приведенными в табл. В.1.

Нет необходимости вводить эти символы в поле Find what самостоятельно, если вам сложно их запомнить. Рядом с окном Find what имеется стрелка, направленная вправо. Щелкните на ней для вызова контекстного меню всех этих полей и щелкните на любом из них для его вставки в окно Find what. (Вам необходимо иметь возможность читать эти символы, чтобы понимать, какое выражение формируется. Кроме того, учтите, что возле поля Find панели инструментов стрелка отсутствует.) Не забудьте установить флажок Regular Expression — только тогда образец будет восприниматься как регулярное выражение.

Ниже приведены примеры регулярных выражений.

- Выражению ^test$ соответствует только строка, состоящая из единственного слова test.
- Выражению doc[1234] соответствует doc1, doc2, doc3 или doc4, но не doc5.
- Выражению doc[1-4] соответствуют те же фрагменты текста, что и выше, но в этом случае само выражение короче.
- Выражению doc[~56] соответствуют doca, doc1 и прочие комбинации, которые начинаются с doc, кроме doc5 и doc6.
- Выражению H\~ello соответствуют Hillo и Hxllo (и многие другие), но не Hello.

- Выражению \{x\!y\}z соответствуют xz и yz.
- Выражению New *York соответствуют New York, а также NewYork и New York.
- Выражению New +York соответствуют New York и New York, но не NewYork.
- Выражению New.*k соответствуют Newk, Newark и New York и многие другие сочетания.
- Выражению \:n соответствуют 0.123, 234, 23.45 (среди других), но не -1C.
- Выражению World$ соответствует World в конце строки, но World\$ соответствует только World$ в любом месте строки.

Таблица В.1. Специальные символы регулярных выражений

Символ	Соответствие
^	Начало строки
$	Конец строки
.	Любой одиночный символ
[]	Любой из символов внутри скобок (символ минуса <⇨ используется для диапазона значений, ^ — для "исключая")
\~	Любой, кроме следующего символа
*	Нуль или больше следующих символов
+	Один или больше следующих символов
{ }	Этот фрагмент искомого текста сохраняется для использования в заменяемом тексте. Таким путем может быть маркировано до девяти фрагментов
\:a	Одиночная буква или цифра
\:b	Непрерывный пробел (несколько следующих подряд табуляций или пробелов)
\:c	Одиночная буква
\:d	Одиночная цифра
\:n	Число без знака
\:z	Целое без знака
\:h	Шестнадцатеричное число
\:i	Строка символов, которая соответствует правилам формирования идентификаторов C++ (начинается с буквы, цифры или символа подчеркивания)
\:w	Строка, состоящая только из букв
\:q	Строковая константа в двойных или одинарных кавычках
\	Возвращает следующему символу его обычный смысл

Edit⇨Find in File

Эта полезная команда используется для одновременного поиска слова или фразы внутри множества файлов. В ее простейшей форме, приведенной на рис. В.25, вы вводите слово или фразу в поле Find what и ограничиваете поиск типами файлов, определенными в поле In files/file types, и каталогом, указанным в поле In folder. В нижней части диалогового окна устанавливаются опции для поиска.

- Match whole word only (Только слово целиком). Если эта опция установлена, то при table в поле Find what в тексте будет найдено только table, но не suitable или tables.

- Match case (Учитывать регистр). Если эта опция установлена, то при Chapter в поле Find what в тексте будет найдено только Chapter, но не CHAPTER или chapter, т.е. регистр букв в образце должен совпадать с регистром в найденном тексте.
- Regular expression (Регулярное выражение). Содержимое поля Find what считается регулярным выражением, если выбрана эта опция.
- Look in subfolders (Просмотр в подкаталогах). Позволяет просмотреть все файлы в подкаталогах заданной папки.
- Output to pane 2 (Вывод в окно 2). Используется для пересылки результатов в ячейку окна Output, связанную с Find in Files 2.

Использование расширенных функций поиска

Справа в нижней части диалогового окна Find in Files есть кнопка Advanced. Щелкнув на ней, можно расширить окно (рис. В.26) и выполнить поиск сразу в нескольких папках.

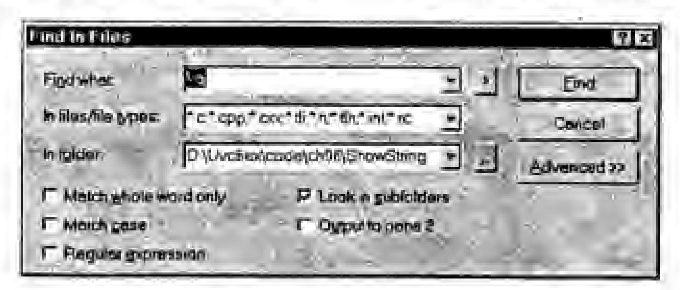


Рис. В.25. Простейший вариант Find in File выполняет поиск заданного текста внутри папки и ее подпапок



Рис. В.26. Кнопка Advanced в окне Find in Files позволяет найти образец в нескольких папках и их подпапках



Если вы выделили текстовый блок до выбора окна Find in Files, этот текст будет помещен в поле Find what. Если выделенного текста нет, в окно Find what попадет слово (или идентификатор), расположенное под курсором.

Результаты работы команды Find in Files появятся на вкладке Find in Files 1 (за исключением того варианта, когда установлена опция Output to Pane 2). Окно Output после этой операции откроется, если оно до сих пор не было выведено на экран. Его размер можно изменить точно так же, как и размеры любого другого окна, "оттянув" мышью элементы рамки. Двойной щелчок на имени файла в списке вывода приводит к открытию этого файла, причем курсор будет установлен на строке, в которой был обнаружен искомый текст.

Edit⇨Replace (<Ctrl+H>)

Эта команда позволяет открыть диалоговое окно Replace, приведенное на рис. В.27. Оно очень похоже на диалоговое окно Find, но используется для замены найденного текста новым. Введите образец в поле Find what и строку замены в поле Replace with. Три флажка — Regular expression, Match case и Match whole word only — имеют то же назначение, что и в диалоговом окне Find. Переключатели группы Replace in (Заменить в...) позволяют при желании ограничить зону поиска/замены выделенным блоком текста.

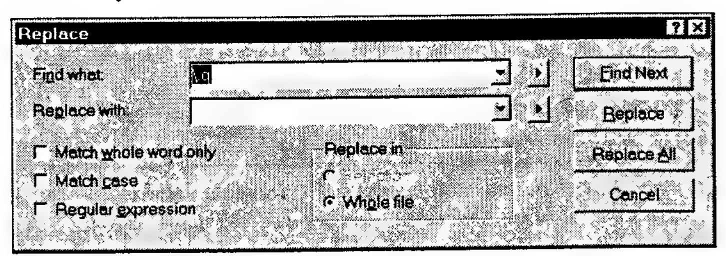


Рис. В.27. Диалоговое окно Replace *используется для замены одного текста другим*

Чтобы увидеть следующий найденный фрагмент перед тем, как дать согласие на его замену, щелкните мышью на Find Next (Найти следующий). Для замены следующего найденного или только что найденного фрагмента щелкните мышью на Replace. Если вы убеждены, что не может быть ни одного случая, когда найденный текст *не нужно* заменять, щелкните на Replace All (Заменять все) — и все фрагменты текста, соответствующие образцу (с учетом установленных опций) будут заменены в файле безо всяких ненужных вопросов. (Для слишком самоуверенных всегда есть Edit⇨UnDo.)

Edit⇨Go To (<Ctrl+G>)

Как на каждом корабле есть центральная рубка, так и в Visual Studio есть диалоговое окно Go To (Перейти к...) (рис. В.28) — центральный пост навигации по файлу. Он позволяет перейти к строке с конкретным номером (по умолчанию), выполнить переход по заданному адресу, ссылке, закладке. Для использования диалогового окна Go To выберите что-нибудь из списка Go to what (Куда идти?). Если выбрано `Line` (Строка), введите номер строки; если выбрано `Bookmark` (Закладка), уберите закладку из списка, и т.д.

В списке Go to what содержатся следующие возможности выбора.

- *Adress* (Адрес). В окнах Memory (Память) и Disassembly (Дезассемблирование) можно перейти к адресу, задаваемому выражением отладчика, как описывается в приложении Г.

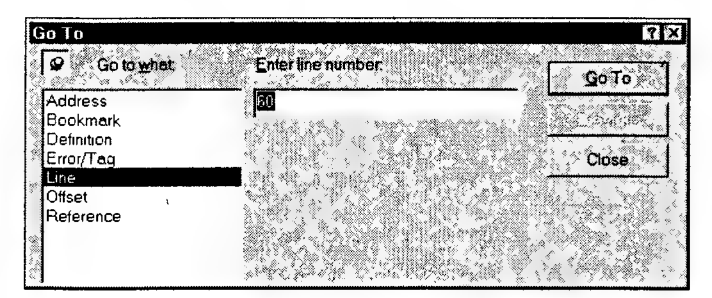


Рис. В.28. Диалоговое окно Go To *позволяет переходить к разным частям проекта*

- *Bookmark* (Закладка). В текстовом файле можно перейти к закладке, хотя разработчикам больше нравится команда Edit⇨Bookmark или выбор пиктограмм, относящихся к закладке, на панели инструментов в той ее части, которая имеет отношение к меню Edit.
- *Definition* (Определение). Если курсор находится на имени функции, открывается исходный (.cpp) файл в том месте, где описан прототип функции. Если курсор расположен на переменной, открывается файл заголовка (.h).
- *Error/Tag* (Ошибки). После завершения компиляции можно переходить от одной найденной компилятором ошибки к другой, дважды щелкнув на строке ошибки в окне Output, обратиться к которому можно из этого диалогового окна. Но удобнее всего просто нажать клавишу <F4>.
- *Line* (Строка). Этот элемент выбран по умолчанию. В поле Enter line number (Введите номер строки) это номер текущей строки.
- *Offset* (Смещение). Вводит смещение адреса (в шестнадцатеричном коде).
- *Reference* (Ссылка). Введите имя (идентификатор функции или объекта) — и курсор будет помещен на строку программы, в которой объявлен этот идентификатор (в разрабатываемой программе или в библиотеке MFC).



Пиктограмма *канцелярская кнопка* в левом верхнем углу этого диалогового окна используется для того, чтобы "приколоть" окно на экране. Тогда оно останется на месте и после перехода в окно файла соответственно запросу. Щелкните на этой пиктограмме, если не нуждаетесь более в услугах окна Go To.

Edit⇨Bookmarks (<Alt+F2>)

Эта команда используется для расстановки закладок в текстовом файле. Диалоговое окно Bookmarks со списком закладок приведено на рис. В.29. Заметим, что временные закладки, установленные с помощью команды окна Find, в этот список не включаются.

Чтобы добавить именованную закладку в строку в окне и запомнить эту строку в файле, введите с клавиатуры в поле Name имя и щелкните на Add. Для перехода к именованной закладке выберите ее из списка и щелкните на Go To. На панели инструментов Edit находятся пиктограммы, которые также можно использовать для добавления (удаления) закладки на то место, где находится курсор, перехода на следующую или предыдущую закладку и удаления всех закладок из файла.

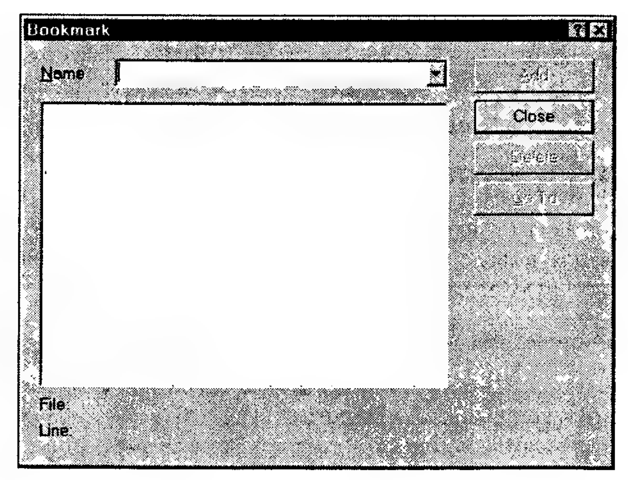


Рис. В.29. Диалоговое окно Bookmarks *управляет закладками, установленными в текстовом файле*

Edit⇨ActiveX Control in HTML

Если инсталлирована система Visual InterDev и вы работаете с элементами управления ActiveX, этот пункт меню позволяет редактировать установки ActiveX. Формирование элементов управления ActiveX обсуждается в главе 17.

Edit⇨HTML Layout

Этот пункт используется для редактирования компоновки HTML с Visual InterDev.

Edit⇨Advanced

Выбор этого пункта приводит к выводу на экран еще одного меню со следующими пунктами.

- Incremental Search (Поиск по приращению). Этот пункт обеспечивает более быстрый поиск, чем тот, который предлагает диалоговое окно Find (о нем шла речь выше). Введите текст образца для поиска в строку состояния. По мере того как вы вводите каждую очередную литеру, Visual Studio ищет текст, который вы к этому моменту ввели. Например, в файле заголовка, если выбрать Edit⇨Advanced⇨Incremental Search и затем ввести p, курсор переместится на первое слово, начинающееся с буквы p (скорее всего, на ключевое слово public). Если затем ввести r, курсор прыгнет на первый pr (скорее всего, на ключевое слово protected). Это избавляет от необходимости ввода полного слова, которое вы хотите отыскать.
- Format Selection (Выбор формата). Этот пункт позволяет установить отступ, используя те же правила, которые применяются при вводе программы.
- Tabify Selection (Табулирование). Преобразует пробелы в табуляции.
- Untabify Selection (Растабулирование). Преобразует символы табуляции в последовательность пробелов.
- Make Selection Uppercase (Перевод в верхний регистр). Преобразует выделенный текст в текст с прописными буквами.

- Make Selection Lowercase (Перевод в нижний регистр). Преобразует выделенный текст в текст со строчными буквами.
- View Whitespace. Вставляет символы-заполннтели (. для пробела и >> для табуляцин), чтобы показать все непрерывные пробелы в документе.

Edit⇨Breakpoints (<Alt+F9>)

Точка останова позволяет приостановить выполненне программы. Команда Edit⇨Breakpoints используется для выаода на экран диалогового окна Breakpoints, показанного на рис. В.30. Подробно работа с этнм окном описана в приложенин Г.

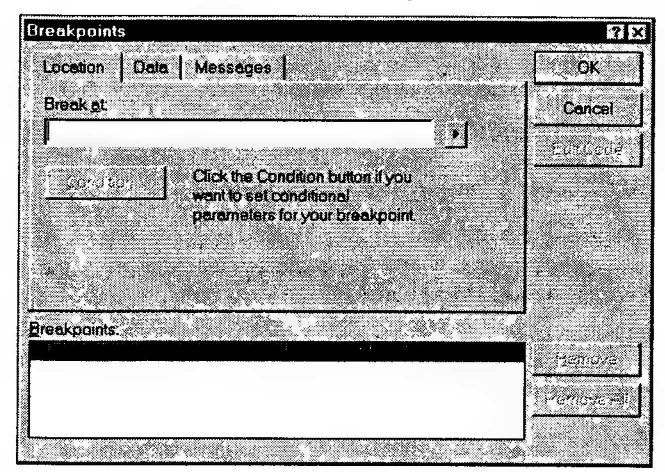


Рис. В.30. Диалоговое окно Breakpoints *используется для отладки приложения*

Edit⇨List Members (<Ctrl+Alt+T>)

Этот пункт меню позволяет активизировать функцию AutoComplete соответственно только что введенному тексту. Открывается список членов класса (переменных и методов), реализация которого редактируется в данный момент. Кроме того, в список включаются все глобальные символы — переменные н функцин. Как правило, этот список довольно внушителен, а потому пользуются им крайне редко.

Edit⇨Type Info (<Ctrl+T>)

Этот пункт меню позволяет вывести контекстное окно с типом переменной. Такое же контекстное окно появится на экране в случае, если указатель мыши задержать на некоторое время на идентификаторе переменной.

Edit⇨Parameter Info (<Ctrl+Shift+клавиша пробела>)

Этот пункт меню позволяет вывести контекстное окно с типами аргументов функции, на которой установлен курсор. Такое же контекстное окно появится на экране в случае, если указатель мыши задержать на некоторое время на идентификаторе функцни.

Edit⇨Complete World(<Ctrl+клавиша пробела>)

Этот пункт меню позволяет активизировать функцию AutoComplete, которая помогает завершить ввод имен функций или переменных, частично введенных с клавиатуры. Если введенный фрагмент имени невелик, т.е. "претендентов" на его завершение достаточно много, открывается диалоговое окно, из которого можно выбрать подходящий идентификатор. Как правило, диалоговое окно AutoComplete появляется сразу же после ввода -> или . с тем, чтобы дать знать пользователю, что нужно ввести имя члена-переменной или метода. Если же функция, которую предполагается вызвать в данном фрагменте текста программы, является методом этого же класса, можно активизировать окно AutoComplete, введя -->.

На заметку Если на вашем компьютере эти команды заблокированы, проверьте настройку функции AutoComplete. Для этого следует выбрать в меню Tools⇨Options и щелкнуть на вкладке Editor, показанной на рис. B.55.

Меню View

Меню просмотра View, показанное на рис. B.31, содержит команды, связанные с внешним видом Visual Studio: какие окна открыты, какие инструментальные средства видимы и т.д.

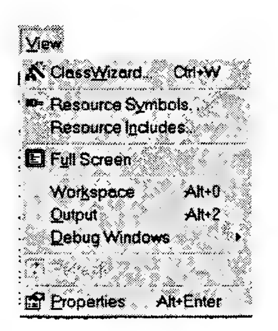


Рис. B.31. Меню просмотра управляет видами Visual Studio

View⇨ScriptWizard

Эта команда, имеющая отношение к InterDev, применяется для редактирования сценариев Web-страниц.

View⇨ClassWizard (<Ctrl+W>)

ClassWizard является, возможно, самым популярным средством в Visual Studio. При добавлении ресурса (меню, диалогового окна, элемента управления и т.д.) вы присоединяете его к программе с помощью ClassWizard. При работе с ActiveX ClassWizard используется для установки свойств, методов и событий. В главе 2 подробно описана методика работы с ClassWizard.

Внимание!

При вызове ClassWizard все измененные файлы сохраняются точно так же, как перед компиляцией. Если вы не желаете сохранять файл после внесения изменений, не вызывайте ClassWizard.

View⇨Resource Symbols

Этот пункт позволяет вывести на экран диалоговое окно Resource Symbols (Символы ресурсов), показанное на рис. B.32. Оно отображает идентификаторы ресурса, такие как ID_EDIT_COPY, используемые в приложении. В верхней части диалогового окна имеется спи-

сок идентификаторов ресурсов, а меньшее окно ниже напоминает назначение ресурса — меню, акселератор, таблица строк и т.д. Кнопки, расположенные справа, применяются для внесения изменений. Для создания нового идентификатора ресурса щелкните на кнопке New (Новый), Delete (Стирание) для удаления идентификатора ресурса (если он не используется), на Change (Замена) для его изменения (если этот идентификатор используется только одним ресурсом) и на кнопке View Use (Просмотр использования) для того, чтобы открыть ресурс (меню, таблицу строк и т.д.), выделенный в нижнем списке.

View⇨Resource Includes

Этот пункт позволяет вывести на экран диалоговое окно Resource Includes, как показано на рис. В.33. Как правило, у вас нет необходимости менять данные в этом окне. В тех редких случаях, когда сгенерированный файл resource.h не совсем вас устраивает, воспользовавшись этим окном, можно добавить в файл дополнительные строки.

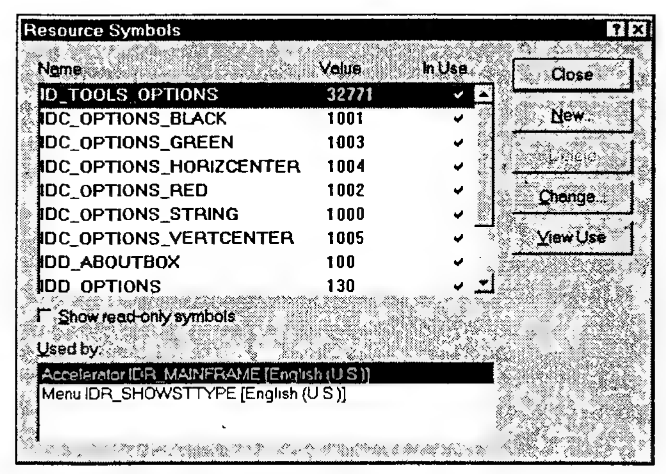


Рис. В.32. В диалоговом окне Resource Symbols *отображаются идентификаторы ресурсов*

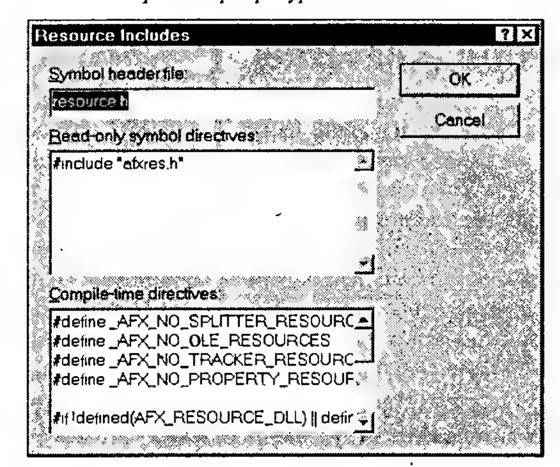


Рис. В.33. Диалоговое окно Resource Includes *позволяет вставлять дополнительные строки директив в файл, описывающий ресурсы проекта*

View⇨Full Screen

Этот пункт позволяет скрыть все панели инструментов, меню, окно вывода и окно Workspace, предоставляя разработчику возможность работать в режиме полного экрана. Остается только кнопка Toggle Full Screen. Для восстановления панели инструментов меню и окон щелкните на этой кнопке.

View⇨Workspace (<Alt+0>)

Выбрав этот пункт, можно вывести на экран окно Workspace, если оно было скрыто. В противном случае команда ничего не делает. Чтобы убрать это окно, щелкните правой кнопкой мыши в окне и выберите Hide или нажмите клавиши <Shift+Esc>, пока окно имеет фокус. На стандартной панели инструментов есть кнопка Workspace, которая позволяет скрыть или отобразить окно.

View⇨Output (<Alt+2>)

Этот пункт позволяет вывести окно Output, если оно скрыто. Чтобы убрать это окно, щелкните правой кнопкой мыши в окне Output, выберите Hide или нажмите клавиши <Shift+Esc>, пока окно имеет фокус. Окно Output появится автоматически, если вы компилируете и компонуете проект или используете Find in Files.

View⇨Debug Windows

Это каскадное меню относится к окнам, используемым во время отладки. Оно обсуждается в приложении Г. Меню содержит следующие пункты.

- Watch (Слежение)
- Call Stack (Стек вызовов)
- Memory (Память)
- Variables (Переменные)
- Registers (Регистры)
- Disassembly (Дезассемблирование).

View⇨Properties (<Alt+Enter>)

Выбор этого пункта приводит к появлению окон свойств с вкладками. Окна свойств для разных элементов отличаются, как показано на рис. В.34–В.36. Эти рисунки иллюстрируют окна свойств для файла исходного текста программы, для таблицы акселераторов, выделенной в окне Workspace, и для одного ключа в этой таблице акселераторов соответственно.

Страницы свойств являются мощным средством редактирования объектов в файлах, не являющихся собственно текстами программ на языке C++, например ресурсов. Однако для функций и переменных обычно проще делать изменения в исходных файлах. Некоторые эффекты становятся доступными только через страницы свойств. Например, страница свойств используется для установки состояния NONE языка, чтобы отменить синтаксическую раскраску файла. (Результат может быть оценен после того, как Windows перерисует окно.)



Обычно окна свойств исчезают, если щелкнуть на каком-либо другом элементе. Если щелкнуть на пиктограмме *канцелярская кнопка* в левом верхнем углу окна, оно фиксируется на экране и отображает свойства всех объектов, с которыми вы работаете.

Рис. В.34. Окно свойств для файла исходного текста программы напоминает вам имя и размер файла, а также разрешает установить язык, используемый для синтаксической раскраски

Рис. В.35. Страница свойств для таблицы акселераторов, в которой устанавливается язык. Можно включать дополнительные таблицы акселераторов в то же самое приложение

Рис. В.36. Страница свойств для элемента таблицы акселераторов предоставляет полный контроль за комбинациями клавиш, ассоциированных с идентификатором ресурса

Меню Insert

В меню Insert, показанном на рис. В.37, собраны команды, относящиеся к вставке чего-нибудь в проект или в один из его файлов.

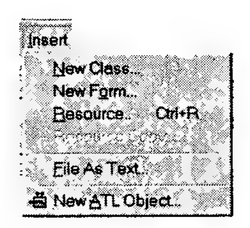


Рис. В.37. Одним из способов добавления элементов в проект или файл является использование меню Insert

Insert⇨New Class

Этот пункт используется для создания файла заголовка или файла исходного текста для нового класса и его добавления в проект. Диалоговое окно New Class показано на рис. В.38. Обратите внимание на раскрывающийся список Base Class, который облегчает спецификацию базового класса.

Insert⇨New Form

При выборе этого пункта меню создается новый класс, производный от `CFormView`, и подключается к приложению. Этот класс может содержать элементы управления, такие как комбинация диалогового окна и представления.

Insert⇨Resource (<Ctrl+R>)

Этот пункт используется для добавления нового ресурса в разрабатываемый проект. Диалоговое окно Insert Resource показано на рис. В.39. Выберите тип ресурса, который должен быть добавлен в проект, и щелкните на OK.

На панели инструментов Resource имеются пиктограммы для добавления нового диалогового окна, меню, курсора, пиктограммы, растра, панели инструментов, акселератора, таблицы строк и информации о версии.

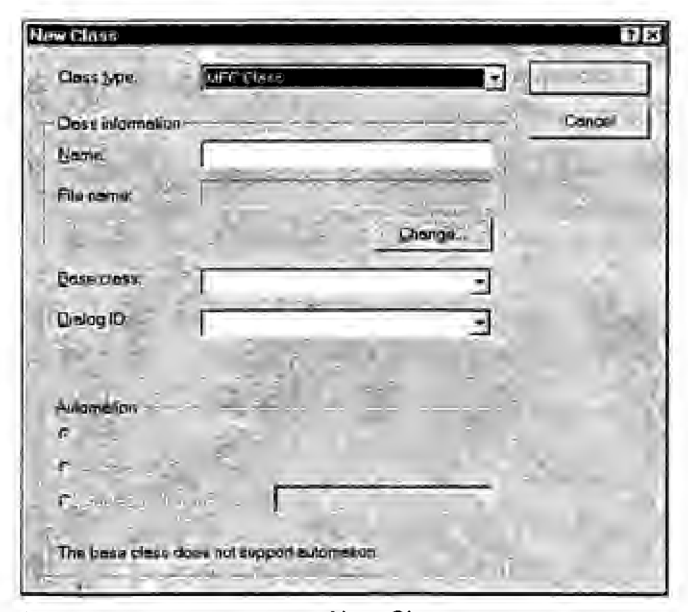


Рис. В.38. Диалоговое окно New Class *упрощает создание новых классов*

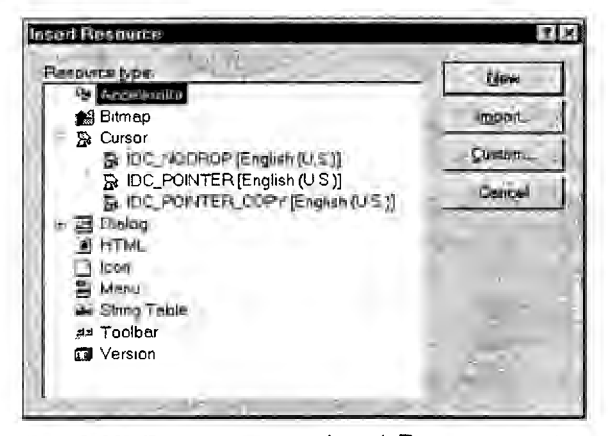


Рис. В.39. Диалоговое окно Insert Resource *— один из инструментов для включения новых ресурсов в проект*

Insert⇨Resource Copy

Используется для копирования имеющихся ресурсов и позволяет изменить только значение в поле Language (Язык), например с американского английского на канадский французский, или Condition (Условие), скажем, для построения отладочной версии диалогового окна. В результате можно создавать несколько версий ресурсов на различных языках, пользуясь директивами компилятора, чтобы определить, какой ресурс будет включен в выполняемый вариант приложения.

Insert⇨File as Text

По этой команде считывается файл с диска и вставляется в редактируемый. Вставка начинается с позиции, в которой находится курсор.

Insert⇨New ATL Object

Этот пункт используется для вставки объектов Active Template Library (ATL) в проект, когда создается элемент управления ActiveX с помощью ATL. Подробнее об этом можно узнать в главе 21.

Меню Project

Меню Project (Проект), приведенное на рис. В.40, содержит пункты, связанные с сопровождением проекта.

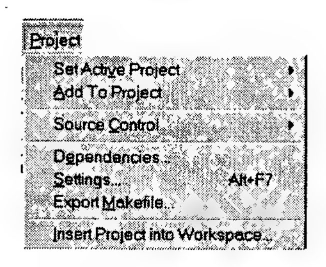


Рис. В.40. Меню Project *упрощает сопровождение проекта*

Project⇨SetActive Project

При наличии нескольких открытых проектов этот пункт позволяет установить, какой из них будет текущим.

Project⇨Add to Project

Этот пункт позволяет вывести на экран каскадное меню со следующими пунктами.

- New (Новый). Выводит на экран такое же диалоговое окно, как и команда File⇨New, причем флажок Add to Project (Добавить в проект) уже установлен.
- New Folder (Новая папка). Создает новую папку для организации классов в проекте.
- Files (Файлы). Выводит на экран диалоговое окно Insert Files Into Project (Вставка файлов в проект), показанное на рис. В.41.
- Data Connection (Подключение данных). Эта команда возможна только в редакции Visual C++ Enterprise Edition. Более подробно она обсуждается в главе 23. Использование этого пункта приводит к объединению проекта с источником данных.
- Components and Controls (Компоненты и элементы управления). Позволяет вывести на экран Components and Controled Gallery, которая обсуждается в главе 25.

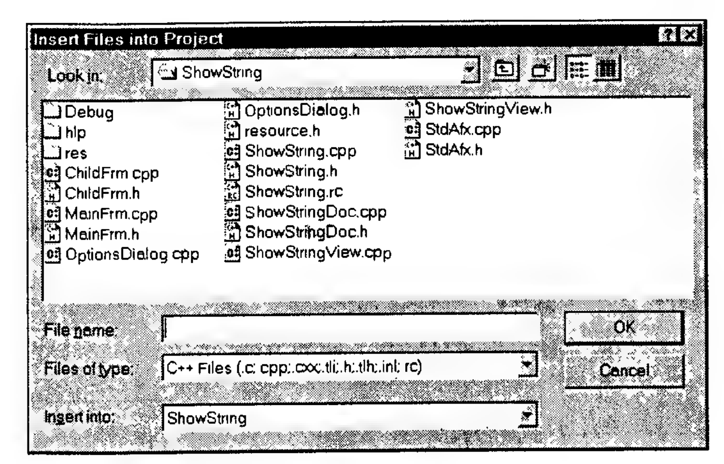


Рис. В.41. Диалоговое окно Insert Files into Project *очень похоже на диалоговое окно* File Open

Project⇨Source Control

По этой команде выполняются задачи отслеживания изменений в файлах программ проекта.

Project⇨Dependencies

Этот пункт позволяет установить зависимость между проектами таким образом, что при изменении одного проекта перекомпилируется и другой, зависящий от первого.

Project⇨Settings (<Alt+F7>)

По этой команде выводится на экран диалоговое окно Project Settimgs, в котором содержатся следующие восемь вкладок.

- General (Общие). Изменяет настройку подключения модулей DLL — статическую, которая была заказана при настройке AppWizard, на разделяемую динамически связываемую — и изменяет каталог, в котором находятся промежуточные (текст программы и классы) или выходные (EXE, DLL, OCX) файлы (рис. В.42).
- Debug (Отладка). Установки этой вкладки обсуждаются в приложении Г.
- C/C++. Это установки компилятора. В списке Category по умолчанию выделен элемент General (Общие). Для изменения установок категории выделите желаемую в этом списке. На рис. В.43 показан вид этой вкладки при выборе категории General. Можно изменить критерий оптимизации в списке Optimization (возможные варианты: Maximize Speed — максимизация скорости, Minimize Size — минимизация размеров, Castomize — по желанию пользователя или Disable — заблокировать, если оптимизация мешает процессу отладки). Кроме того, можно изменить уровень предупреждающих сообщений компилятора в списке Warning level. Эта вкладка обсуждается более детально в главе 24.
- Link (Компоновка). Эта вкладка управляет опциями компоновки, которые вам потребуется заменить (что маловероятно). Установки разделены на пять категорий. Категория General приведена на рис. В.44.
- Resources (Ресурсы). Эта вкладка, показанная на рис. В.45, используется для изменения языка, на котором написаны тексты ресурсов. Она разрешает заменить ресурсы, которые компилируются в приложении, и другие установки ресурсов.

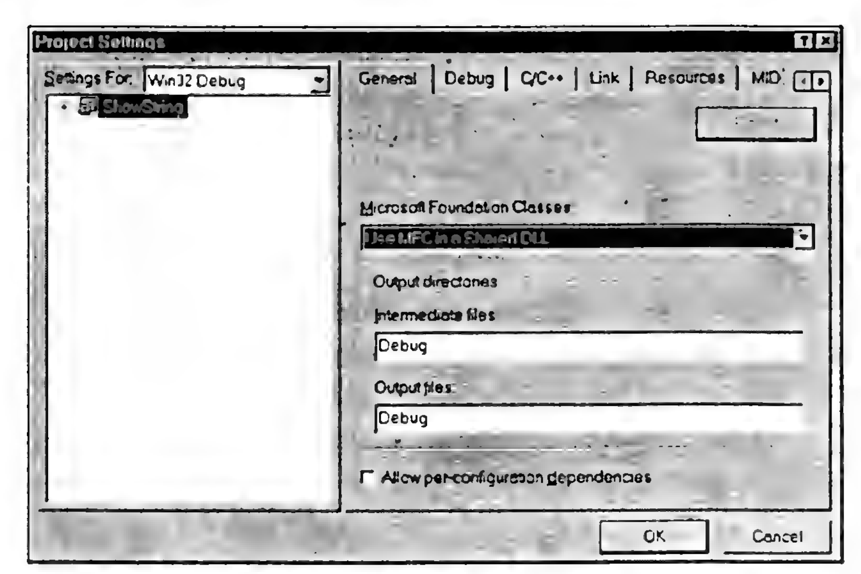


Рис. В.42. Вкладка General *диалогового окна* Project Settings *позволяет управлять распределением файлов по каталогам*

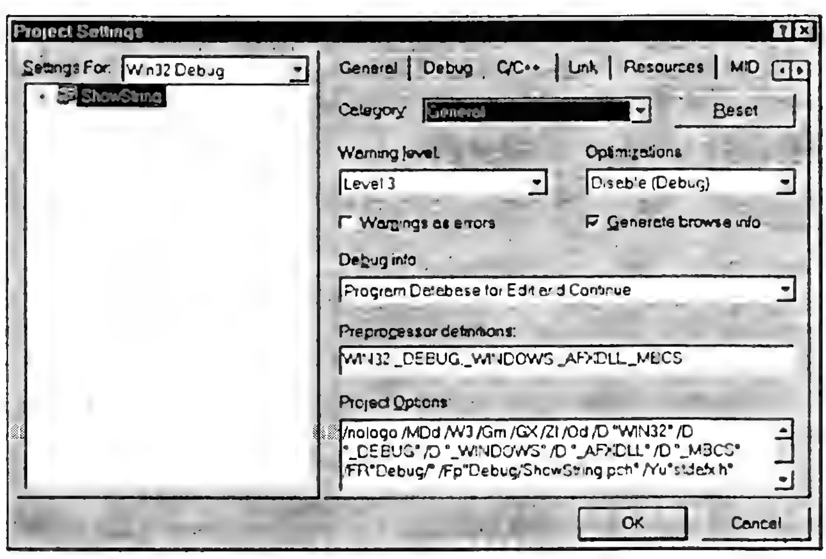


Рис. В.43. Вкладка C/C++ *диалогового окна* Project Settings *позволяет управлять установками компилятора по восьми категориям, начиная с* General

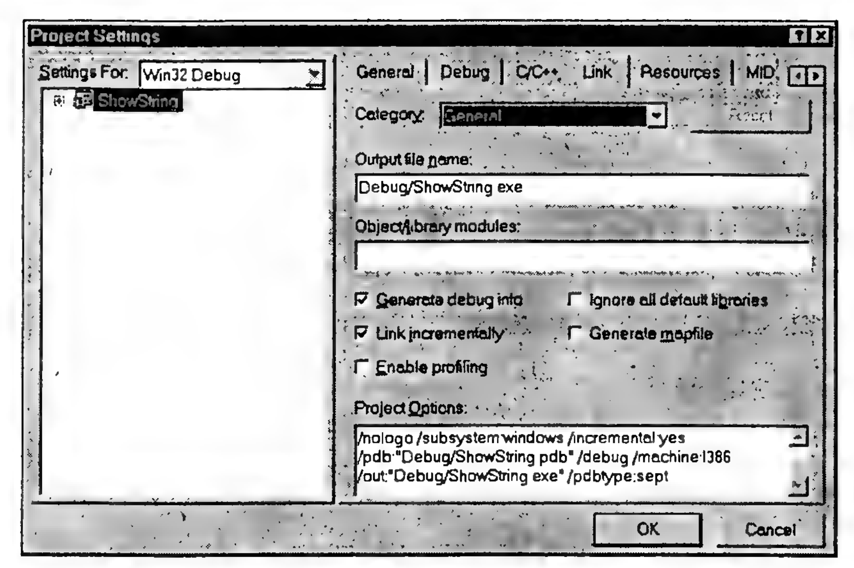


Рис. B.44. Вкладка Link *диалогового окна* Project Settings *позволяет управлять установками компоновщика по пяти категориям, начиная с* General

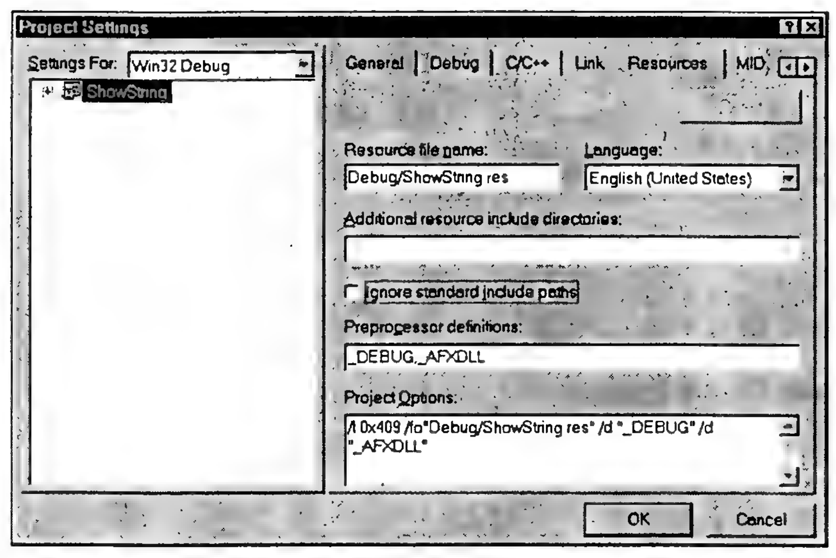


Рис. B.45. Вкладка Resources *диалогового окна* Project Settings *позволяет управлять установками ресурсов, включая язык*

- MIDL (Типы OLE). Эта вкладка используется теми программистами, которые строят библиотеку типов (TLB) из файла описания объектов (.ODL). Файлы .ODL обсуждаются в главе 16.
- Browse Info (Информация просмотра). Эта вкладка, приведенная на рис. B.46, управляет файлом Browse Info (.bsc), используемым для пунктов меню Go To Definition, Go To Declaration и аналогичных им. Если у вас нет необходимости в этих функциях, можете повысить скорость компоновки, отключив генерацию информации для просмотра. Если же вы хотите просмотреть информацию, то в дополнение к флажку Build

browse info file на этой вкладке уствновите еще и флажок Generate browse info в категории General вкладки C/C++.

- Custom Build. Эти установки позволяют вам добавлять собственные этапы как часть каждого процесса компиляции/компоновки приложения.
- Pre–Link Step. Вы можете добавить собственные этапы непосредственно перед этапом компоновки.
- Post–Build Step. Вы можете добавить собственные этапы после того, как все этапы стандартной процедуры компиляции/компоновки приложения будут успешно завершены.

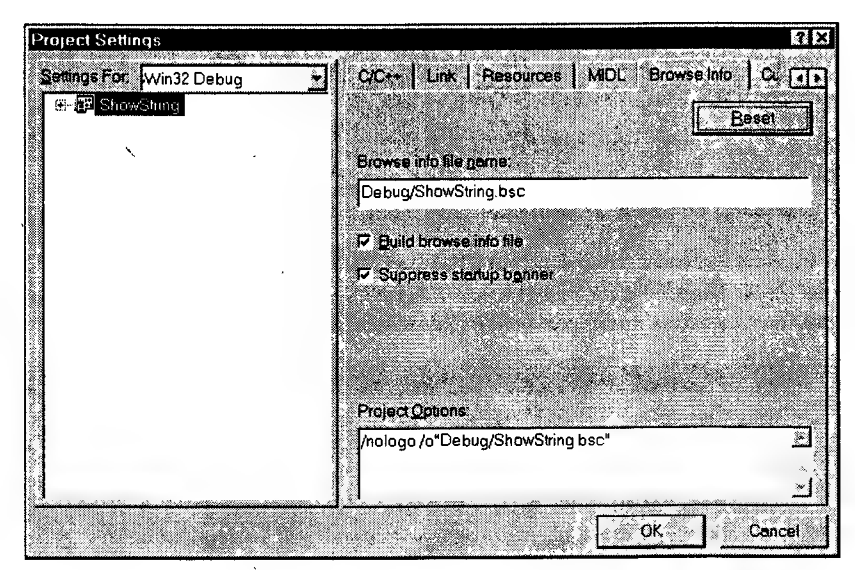


Рис. В.46. Вкладка Browse Info *диалогового окна* Project Settings *позволяет включать или выключать функции просмотра*

Чтобы просмотреть несколько последних вкладок, щелкните правой кнопкой мыши в конце списка вкладок. Вы можете сделать установки для любой конфигурации (Debug (отладочная), Release (распространяемая) и т.д.) по отдельности или вместе. Многие подокна имеют кнопку Reset, с помощью которой восстанавливаются параметры того состояния, в котором они были при первоначальной настройке проекта.

Меню Build

Меню Build, приведенное на рис. В.47, включает все команды, связанные с компиляцией, выполнением и отладкой приложений.

Именно в этом меню сосредоточивается основная часть операций после того, как вы готовы к компиляции и отладке. Меню Build включает следующие пункты.

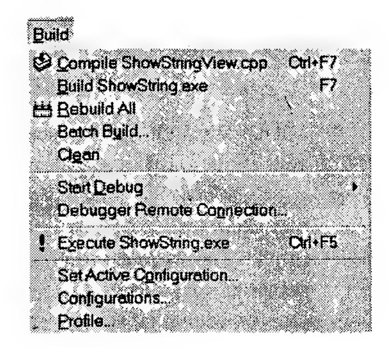


Рис. В.47. Меню Build *используется для компиляции, компоновки и отладки приложений*

Build⇨Compile (<Alt+F7>)

При выборе этого пункта компилируется файл, на имени которого находится фокус. Это очень полезная функция, если вас волнует поиск ошибок, особенно во время первой компиляции после внесения исправлений в программе. Например, если есть ошибка в файле заголовка, включенном в множество исходных файлов, компилятор формирует сообщения об ошибках, относящихся к этому файлу заголовка каждый раз, когда компилируется очередной исходный файл. При наличии предупреждений в одном из исходных файлов проект компонуется, но вы, возможно, пожелаете остановить процесс компоновки и внести соответствующие исправления. На панели инструментов Project есть пиктограмма Compile в виде стопки листов со стрелкой, направленной вниз.

Build⇨Build (<F7>)

С помощью этой команды компилируются и компонуются все измененные в проекте файлы. На панели инструментов Project есть пиктограмма Build.

Build⇨Rebuild All

По этой команде компилируются и компонуются *все* файлы в проекте (даже те, которые не были изменены). Иногда в процессе повторной компиляции и компоновки пропускается файл, который должен быть перекомпилирован, и эта команда позволяет исправить ситуацию.

Build⇨Batch Build

Проект содержит, как минимум, две *конфигурации*: `Debug` (отладочная) и `Release` (распространяемая). Обычно вы работаете с конфигурацией `Debug`, изменяя, строя, проверяя, снова изменяя проект до тех пор, пока он не будет отлажен. После этого строится версия `Release`. Если возникает необходимость в создании нескольких конфигураций одновременно, этот пункт меню используется для вывода на экран диалогового окна Batch Build, показанного на рис. B.48. Для компиляции только измененных файлов выберите Build, а для компиляции всех файлов — Rebuild All. За успешной компиляцией последует компоновка файлов.

Build⇨Clean

Этот пункт позволяет удалить все временные файлы и файлы вывода, так что в проекте остаются только исходные файлы.

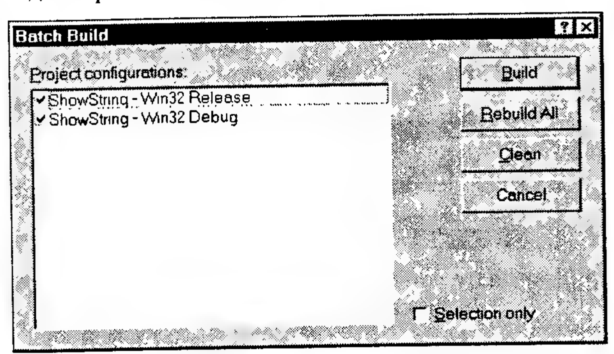


Рис. B.48. Диалоговое окно Batch Build позволяет создать несколько конфигураций проекта одновременно

Build⇨Start Debug

Отладка обсуждается в приложении Г.

Build⇨Debugger Remote Connection

Можно выполнять программу на одном компьютере, а отлаживать — на другом. Этот пункт используется как часть процесса удаленной отладки для связи двух компьютеров. Процесс удаленной отладки обсуждается в приложении Г.

Build⇨Execute (<Ctrl+F5>)

Если выбрать Build⇨Execute, эта команда запускает выполнение приложения без вызова на экран отладчика.

Build⇨Set Active Configuration

Диалоговое окно Set Active Configuration, показанное на рис. В.49, устанавливает, какая из конфигураций является активной (обычно это Debug или Release). Активная конфигурация проекта создается с помощью команд Build.

Build⇨Configurations

Выбор этого пункта приводит к появлению диалогового окна Configurations, показанного на рис. В.50. В этом окне можно добавить новую конфигурацию либо удалить прежнюю. Чтобы изменить установки для новых конфигураций, используется команда Project⇨Settings.

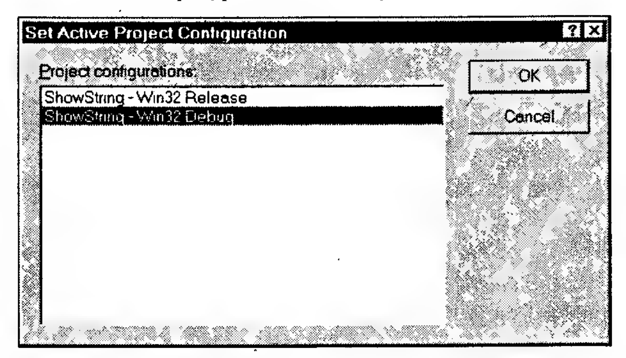


Рис. В.49. В диалоговом окне Set Active Configuration *установлена конфигурация по умолчанию*

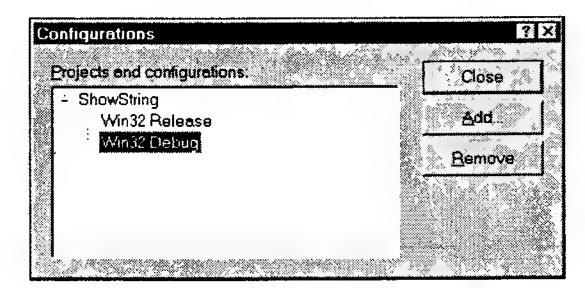


Рис. В.50. Диалоговое окно Configurations *позволяет добавлять конфигурацию к стандартным* Debug *и* Release

Build⇨Profiler

Система построения профиля программы является мощным средством определения "узких мест" в вашем приложении. Этот пункт обсуждается в главе 24.

Меню Tools

Меню Tools, приведенное на рис. В.51, упрощает доступ к инструментальным средствам с расширенными возможностями и хранит команды, которые не были включены в остальные меню.

Tools⇨Source Browser (<Alt+F12>)

Окно просмотра позволяет значительно расширить возможности Visual Studio. Оно используется каждый раз, когда вы обращаетесь к определению или ссылке, анализируете граф вызовов или исследуете другим способом связи между классами, функциями и категориями в проекте. Однако доступ к окну просмотра через этот пункт меню выглядит необычно. При этом на экран вызывается диалоговое окно Browse, показанное на рис. В.52. Возможно, вам больше по душе команда Edit⇨Go To или одна из одиннадцати пиктограмм панели инструментов Browse.

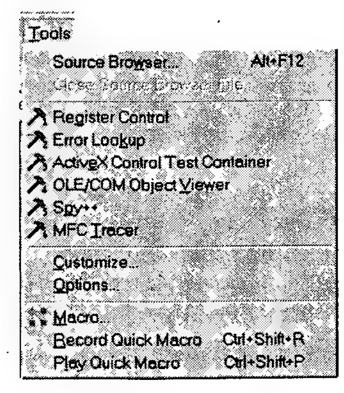


Рис. В.51. Меню Tools позволяет организовать расширение инструментальных средств

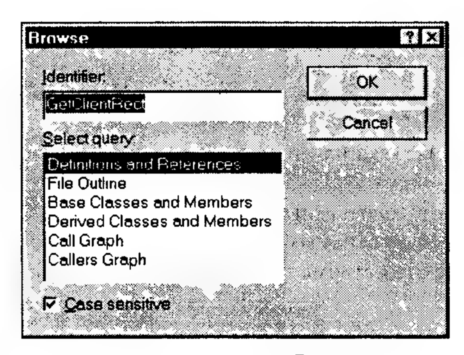


Рис. В.52. Диалоговое окно Browse — кратчайший способ вызова процедуры просмотра объектов, функций и переменных

Tools⇨Close Source Browser File

Файл просмотра перестраивается вместе с проектом. Если вы внесли изменения в свой проект вне Visual Studio, используя средство наподобие NMAKE, вы должны сначала закрыть файл просмотра, выбрав эту команду, чтобы его можно было обновить.

Дополнительные инструменты

При инсталляции Visual C++ в меню Tools включается несколько пунктов. Дополнительные средства можно добавить с помощью пункта меню Customize.

Tools⇨Customize

Диалоговое окно Customize появляется при выборе Tools⇨Customize. Вкладка Commands этого окна показана на рис. В.53. Одиннадцать пиктограмм в группе Buttons относятся к командам меню File, и, если вы хотите установить пиктограмму одной из этих команд на

некоторой панели инструментов, просто перетащите ее из группы Buttons на подходящее место в выбранной панели инструментов и отпустите кнопку мыши. Помните, что в этом случае панель инструментов служит и окном меню.

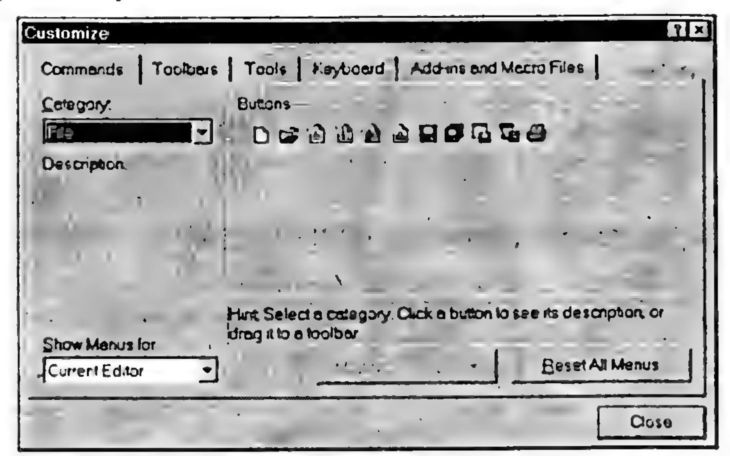


Рис. В.53. Вкладка Commands *диалогового окна* Customize *позволяет вам формировать собственные панели инструментов*



Если в результате слишком тщательной модернизации все панели инструментов пришли в негодность (в них слишком много дополнительных пиктограмм или очень много пиктограмм удалено), то кнопка Reset All Menus (Восстановить все меню) возвратит объекты в их исходное состояние.

Вкладка панели инструментов, показанная на рис. В.54, — это один из инструментов контроля за тем, какая панель инструментов отображается. Очевидно, что вы можете запретить вывод контекстных окон указателя (ToolTips), если они вас не устраивают, перейти на большие пиктограммы, если есть для них место. (На рис. В.54 показана настройка стандартной панели инструментов для использования больших пиктограмм.)

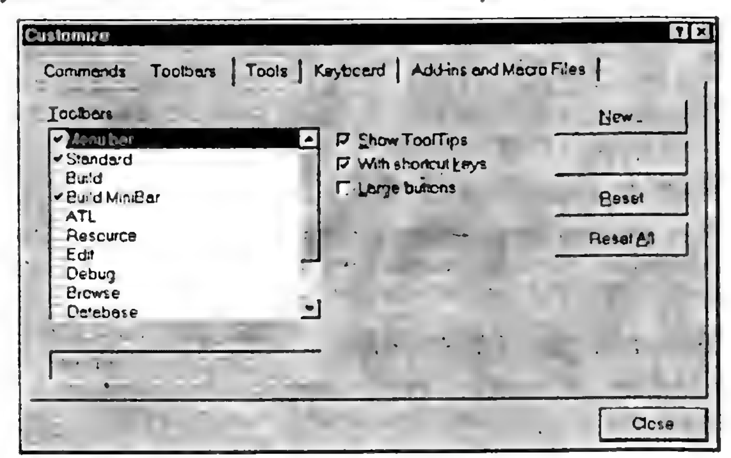


Рис. В.54. Вкладка Toolbars *диалогового окна* Customize *представляет собой одно из средств включения или выключения панелей инструментов и единственное средство управления форматом контекстного окна указателя и размерами пиктограмм*

Вкладка Tools позволяет добавлять программы в меню Tools, а вкладка Keyboard — менять комбинации клавиш ускоренного вызова команд или связывать новые комбинации с командами, которые ранее обходились без этого. Вкладки Add-Ins и MacroFiles предоставляют возможность добавлять *макросы*, написанные на VBScript и позволяющие автоматизировать многие задачи Visual Studio, или *вставки (add-ins)*, которые могут быть написаны на любом языке и также автоматизировать работу Visual Studio.

Tools⇨Options

Этот пункт содержит большое число установок и опций, которые относятся к Visual Studio. Например, на рис. B.55 приведена вкладка Editor диалогового окна Options. Если в Visual Studio есть опция, установка которой вас не устраивает, можете почти наверняка заменить ее настройку с помощью этого большого окна.

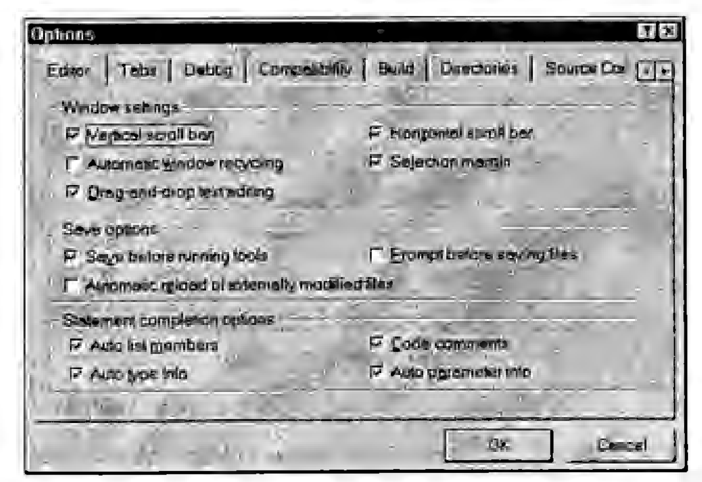


Рис. B.55. Вкладка Editor диалогового окна Options — это место, где можно изменить установки редактора

Вкладки этого окна перечислены ниже.

- Editor. Выбирает полосу прокрутки, разрешает операцию "перетащить и опустить" (drag-and-drop), устанавливает автоматическую запись и загрузку.
- Tabs. Устанавливает опции, относящиеся к символам табуляции (вставленным при нажатии клавиши <Tab>) и отступам (вставляются редактором в новые строки после элементов языка C++ наподобие фигурных скобок).
- Dialog. Определяет, какая информация отображается в процессе отладки.
- Compatibility. Позволяет эмулировать другой редактор (Brief либо Epsilon) или хотя бы некоторую часть его интерфейса.
- Build. Генерирует внешний файл управления компиляцией/компоновкой или составляет список файлов для компиляции/компоновки.
- Directories. Устанавливает каталоги, в которых находятся файлы заголовков, выполняемые, библиотечные и исходные файлы.
- Source Control. Устанавливает опции, имеющие отношение к Visual SourceSafe, о котором шла речь в главе 23.
- Workspace. Устанавливает стационарные окна, строки состояния и порядок загрузки файлов проекта (рис. B.56).

- **Data View**. Управляет выводом на экран окна просмотра данных DataView (только в редакции Visual C++ Enterprise Edition).
- **Macros**. Устанавливает правила перезагрузки измененного макроса.
- **Help System**. Определяет источник информации для системы электронной справки (как правило, MSDN).
- **Format**. Устанавливает схему раскраски, включая схему синтаксической раскраски для исходных файлов.

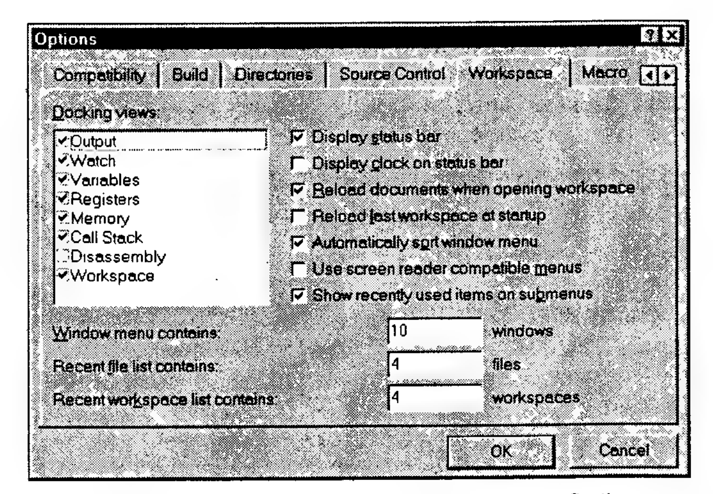


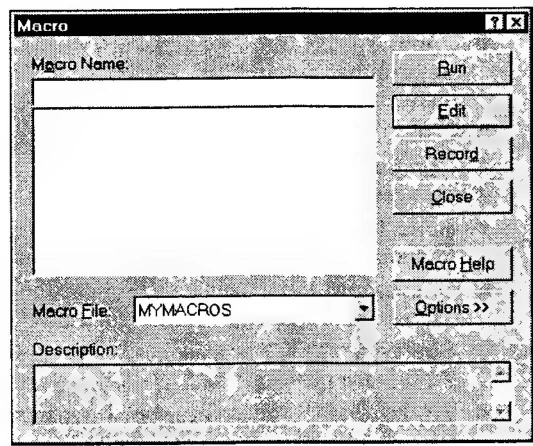
Рис. В.56. Вкладка Workspace диалогового окна Options позволяет установить, какие окна будут стационарными, какие плавающими, а также опции управления порядком загрузки



Если вы работаете продолжительное время над одним и тем же проектом, установите флажок Reload last workspace at startup на вкладке Workspace. В этом случае при запуске Visual Studio будет автоматически загружаться тот проект, над которым вы работали в предыдущем сеансе.

Tools⇨Macro

Этот пункт позволяет вывести на экран диалоговое окно Macro, показанное на рис. В.57. С его помощью можно записать или воспроизвести простой макрос, или отредактировать набор записанных нажатий клавиш, добавляя в них операторы VBScript.



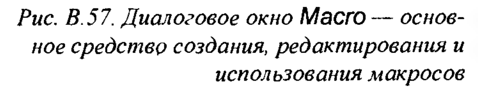


Рис. В.57. Диалоговое окно Macro — основное средство создания, редактирования и использования макросов

Tools⇨Record Quick Macro

Если вы не собираетесь создавать именованный макрос с тем, чтобы использовать его в разных проектах, а хотите только на ходу создать подсобный инструмент такого рода для текущей задачи, запишите "быстрый" макрос при помощи этой команды. Такой макрос может быть только один. При записи нового макроса прежний макрос уничтожается.

Tools⇨Play Quick Macro

Вызывает отработку последнего записанного "быстрого" макроса.

Меню Window

Меню Window (рис. В.58) управляет окнами в основной рабочей зоне Visual Studio.

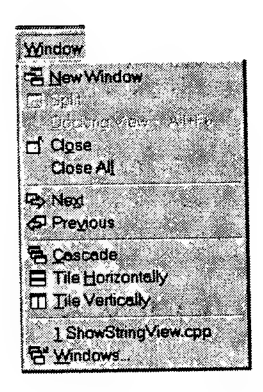


Рис. В.58. Меню Window позволяет управлять окнами Visual Studio в основной рабочей зоне

Window⇨New Window

Выбрав этот пункт, можно открыть другое окно как окно с фокусом с тем же исходным файлом. Строка заголовка первого окна изменяется — к имени файла добавилось :1; во втором окне к имени файла добавилось :2. Изменения, сделанные в первом окне, немедленно отражаются во втором окне. Окна можно протягивать и закрывать, а также изменять их размеры независимо друг от друга.

Window⇨Split

Эта команда позволяет разделить окно, в котором расположен фокус; когда вы щелкнете кнопкой мыши, окно разделится на четыре подокна вдоль линий перекрестия. Эти границы можно перемещать обычным путем, если они вас не устраивают. Прокрутка одного подокна приводит к прокрутке и подокна-компаньона. Для отмены деления окна переместите правую границу по направлению к углу окна, и она исчезнет. Переместите горизонтальную и вертикальную границы, и окно больше не будет разделено на ячейки.

Window⇨Docking View (<Alt+F6>)

Это команда управления видом окна с фокусом — является ли оно стационарным или плавающим. Пункт меню блокируется, если фокус находится в основной рабочей зоне.

Window⇨Close

Этот пункт позволяет закрыть окно с фокусом и связанный с ним файл. Если у вас есть не сохраненные изменения в файле, поступает запрос на их сохранение.

Window⇨Close All

Этот пункт позволяет закрыть все окна в основной рабочей зоне. Если у вас есть не сохраненные файлы, поступает запрос на их сохранение.

Window⇨Next (<Ctrl+Tab>)

Этот пункт позволяет переключить фокус на следующее окно. Порядок окон может быть определен во время просмотра списка открытых окон в нижней части меню.

Window⇨Previous (<Ctrl+Shift+Tab>)

Этот пункт позволяет переключить фокус в предыдущее окно.

Window⇨Cascade

Этот пункт позволяет упорядочить все окна в основной рабочей зоне в соответствии со стандартным каскадным форматом, как показано на рис. В.59. Окна, свернутые в пиктограммы, не восстанавливаются и не включаются в каскад.

Window⇨Tile Horizontally

Этот пункт позволяет расставить окна в основной рабочей зоне таким образом, что каждое из них занимает всю ширину рабочей зоны, как показано на рис. В.60. При выборе этого пункта файлы с фокусом располагаются сверху.

Window⇨Tile Vertically

Этот пункт позволяет расставить окна в основной рабочей зоне таким образом, что каждое из них занимает всю высоту рабочей области, как показано на рис. В.61. При выборе этого пункта файлы с фокусом располагаются слева.

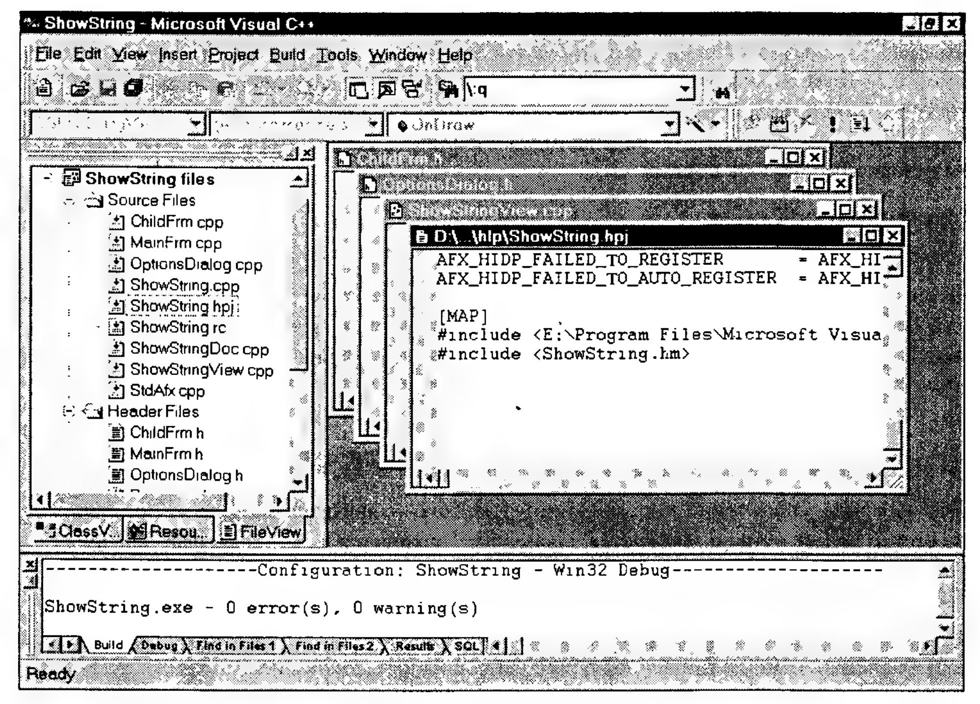


Рис. В.59. Расположив окна каскадом, можно упростить переключение между ними

Рис. В.60. Когда окна стыкуются по горизонтали, каждое из них занимает всю ширину рабочей области

Рис. В.61. Когда окна стыкуются по вертикали, каждое из них занимает всю высоту рабочей области

Список открытых окон

В нижней части меню содержится список окон в основной рабочей зоне, так что можно перемещаться между ними даже в случае, когда окна распахнуты на всю рабочую область. Если открытых окон больше девяти, в списке будут только первые девять. К остальным можно получить доступ, выбрав Window⇨Windows.

Window⇨Windows

С помощью этого пункта можно вывести в рабочей зоне диалоговое окно Windows (рис. В.62). Оно позволяет закрыть, сохранить или активизировать любое окно.

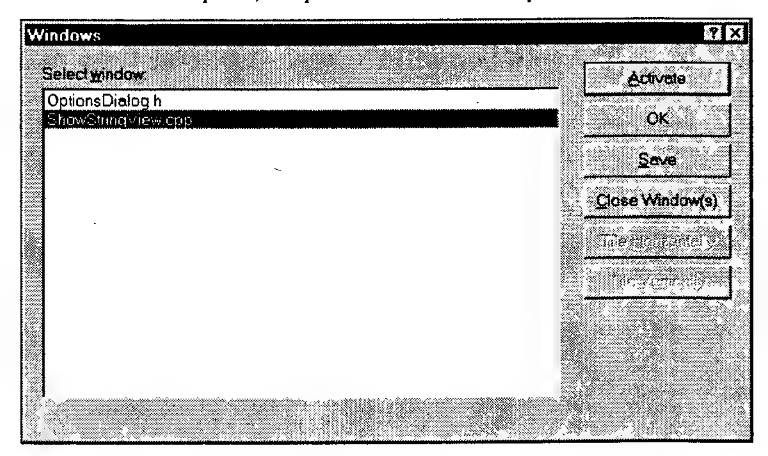


Рис. В.62. Диалоговое окно Windows предоставляет доступ к любому окну в основной рабочей области

Меню Help

В Visual Studio справочная система является отдельным программным продуктом. В большинстве диалоговых окон нажатие клавиши <F1> вызывает на экран стандартную справочную систему, так же как выбор пунктов меню (рис. В.63).

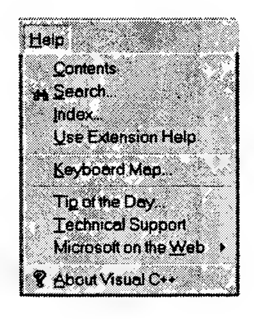


Рис. В.63. Меню Help — средство доступа к системе электронной справки

Help⇨Contents

Этот пункт позволяет запустить программный продукт MSDN, если последний не был запущен ранее, и вывести на экран вкладку Table of Contents (Содержание).

Help⇨Search

Этот пункт позволяет открыть вкладку Search продукта MSDN.

Help⇨Index

Этот пункт позволяет открыть вкладку Index продукта MSDN.

Help⇨Use Extention Help

При вызове этого пункта переключается источник справочной информации — от продукта MSDN к альтернативному, установленному пользователем самостоятельно, или наоборот.

Help⇨Readme

Вывод на экран файла read me для Visual C++.

Help⇨Keyboard Map

Этот элемент не включен в InfoViewer. Его выбор приводит к появлению диалогового окна Help Keyboard (Справка, клавиатура), приведенного на рис. B.64. В верхней части этого диалогового окна имеется раскрывающийся список, предназначенный для выбора групп команд, комбинации клавиш для которых вы хотите просмотреть: Bound — команды, которым присвоены комбинации клавиш, All — все команды, наименования меню команд File, Edit, View, Insert, Build, Debug, Tools, Window и Help. Кроме того, доступны команды, относящиеся к Images и Layout.

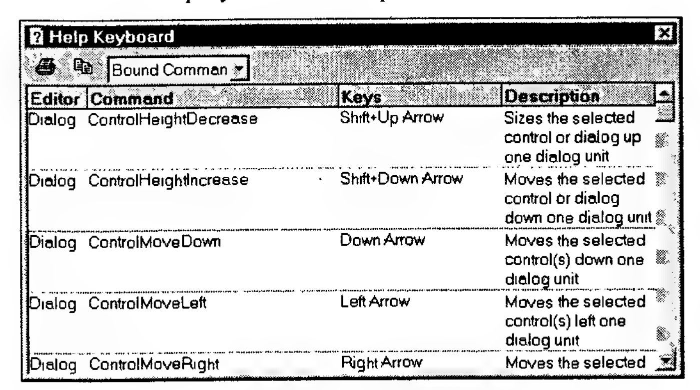
Для сортировки столбцов щелкните на их заголовках в верхней части таблицы. Если необходимо изменить комбинацию клавиш для соответствующих команд, выберите Tools⇨Customize и используйте вкладку Keyboard.

Help⇨Tip of the Day

Выбрав этот элемент, можно вывести на экран окно Tip of the Day (Памятки на каждый день), как показано на рис. B.65. Одни из них являются памятками о секретах Windows, другие относятся к Visual Studio. Если всякий раз, когда открывается Visual Studio, вы не можете ждать, пока появится новая памятка, щелкните на Next Tip для прокрутки списка. Если вам досаждают эти памятки, с самого начала отмените выбор Show Tips (Отображать памятки) в окне Startup.

Help⇨Technical Support

Используется для технической поддержки. Выбрав этот пункт, можно получить не только поддержку, но и ответы на интересующие вас вопросы.



Editor	Command	Keys	Description
Dialog	ControlHeightDecrease	Shift+Up Arrow	Sizes the selected control or dialog up one dialog unit
Dialog	ControlHeightIncrease	Shift+Down Arrow	Moves the selected control or dialog down one dialog unit
Dialog	ControlMoveDown	Down Arrow	Moves the selected control(s) down one dialog unit
Dialog	ControlMoveLeft	Left Arrow	Moves the selected control(s) left one dialog unit
Dialog	ControlMoveRight	Right Arrow	Moves the selected

Рис. B.64. В диалоговом окне Help Keyboard *отображаются комбинации клавиш, ассоциированных с командами*

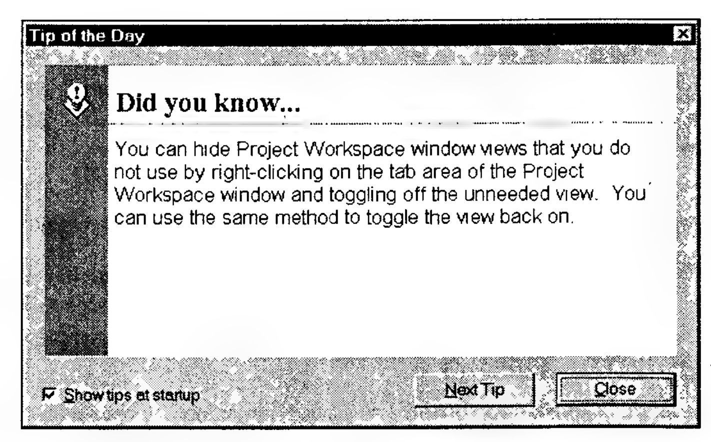


Рис. В.65. Окно Tip of the Day *предоставляет большие возможности для подробного изучения Visual Studio*

Help⇨Microsoft on the Web

Один из путей получения информации о Visual Studio и других средствах — использование World Wide Web. Выбор этого пункта приводит к появлению каскадного меню со списком Web-страниц. Выбор любого пункта этого меню приводит к отображению на экране соответствующей страницы по умолчанию в окне InfoViewer. Можете при желании организовать запуск другого окна просмотра: выберите Tools⇨Options, щелкните на вкладке InfoViewer и отмените выбор Use Infoviewer for Microsoft on the Web. После этого будет по умолчанию использоваться ваше окно просмотра.

Help⇨About Visual C++

Выбор этого элемента приводит к появлению окна About для Visual Studio, которое включает, помимо другой информации, ваш идентификатор продукта.

Панели инструментов

После того как вы хорошо освоите процедуры Visual Studio, панели инструментов сэкономят много вашего времени. Вместо выбора File⇨Open, требующего двух щелчков и движения мыши, проще щелкнуть на пиктограмме панели инструментов. Однако в данном продукте имеется одиннадцать панелей инструментов и панель меню, поэтому придется запомнить множество пиктограмм. В этом разделе приводятся сведения обо всех панелях инструментов и соответствии между элементами меню и пиктограммами.

На рис. В.66 приведены все панели инструментов, имеющиеся в Visual Studio. Быстрее всего можно вызвать на экран или удалить панель инструментов с помощью диалогового окна Toolbars, которое, кроме того, может использоваться для включения и выключения контекстного окна указателя (ToolTip), а также для того, чтобы установить, будет ли текст в нем содержать сведения о комбинации клавиш ускоренного вызова команды. Любая из этих панелей может быть зафиксирована на любой из границ рабочей области, как показано на рис. В.67. Для перемещения зафиксированной панели инструментов перетащите ее за *бордюр* (wrinkles) — две вертикальные полоски по краям. Плавающие панели инструмеитов перемещаются по экрану так же, как и любые другие окна. Когда панель приближается к границе рабочей области, изменение ее формы показывает, что панель будет пристыкована к границе. Проведите несколько экспериментов с перемещением панелей инструментов, пока не найдете конфигурацию, которая вас устроит.

Рис. В.66. Visual Studio содержит одиннадцать панелей инструментов

Рис. В.67. Панели инструментов Visual Studio могут быть пристыкованы к любой границе

Две самые важные панели инструментов — Standard (Стандартная) и Build (Компиляция/компоновка). Для полного описания функций каждой пиктограммы необходимо возвратиться к рассмотренным выше в этом приложении соответствующим элементам меню.

Стандартная панель инструментов

Стандартная панель инструментов помогает поддерживать и редактировать в рабочей области текст и файлы. В табл. В.2 приведены названия каждой стандартной пиктограммы и эквивалентной ей команды меню.

Таблица В.2. Пиктограммы стандартной панели инструментов и эквивалентные команды меню

Название пиктограммы	Команда меню
New Text File	File⇨New
Open	File⇨Open
Save	File⇨Save
Save All	File⇨Save All
Cut	Edit⇨Cut
Copy	Edit⇨Copy
Paste	Edit⇨Paste
Undo	Edit⇨Undo
Redo	Edit⇨Redo
Workspace	View⇨Workspace
Output	View⇨Output
Window List	Window⇨Windows
Find in Files	Edit⇨Find in Files
Search	Help⇨Search

Панель инструментов Build

В табл. В.3 определены пиктограммы панели инструментов Build, относящиеся к компиляции и отладке.

Таблица В.3. Пиктограммы панели инструментов Build и эквивалентные команды меню

Название пиктограммы	Команда меню
Compile	Build⇨Compile
Build	Build⇨Build
Stop Build	Build⇨Stop Build
Execute	Build⇨Execute
Go	Build⇨Start Debug⇨Go
Insert/Remove Breakpoint	Отсутствует

Применение других панелей инструментов

Вы можете вывести на экран любую панель инструментов, добавить на нее или убрать с нее пиктограммы и добиться того, чтобы приложение Visual Studio работало именно так, как вам удобно. Проведите эксперименты и посмотрите, как это упрощает процесс программирования.

ПРИЛОЖЕНИЕ Г

Отладка

В этом приложении...

Терминология отладки

Команды и окна отладки

Применение утилиты MFC Tracer

Метод Dump()

Отладка является существенной частью процесса программирования. Если программа выполняется до конца, но при этом делает не то, что для нее планировалось, нужно выяснить, что же происходит на самом деле. Для этого придется обратиться к отладчику. Некоторые философские и технические аспекты отладки изложены в разных главах настоящей книги, особенно в главе 24. В этом приложении приводится описание основных средств, используемых в процессе отладки: меню, панелей инструментов, окон, о которых не рассказывалось в приложении В.

Терминология отладки

Очевидно, что при отладке программ ключевым понятием является термин *точка останова.* Точка останова — это место в программе, в котором вы хотите остановиться. Возможно, вас заинтересует количество выполняемых циклов или место, в которое передается управление внутри оператора if, или место, откуда происходит вызов функции. В точке останова останавливается выполнение программы перед выполнением соответствующего оператора программы. В этом месте вы можете прекратить выполнение программы и перезапустить ее или работать дальше и т.д. Можно также проанализировать значения некоторых переменных или просмотреть стек вызовов и увидеть, каким образом управление передается на этот оператор.

Чтобы продолжить выполнение программы, используйте следующие команды.

- *Go* — выполнять до следующей точки останова или до конца программы, если далее не встретятся точки останова.
- *Restart* — возобновить выполнение программы с самого начала.
- *Step Over* — выполнить только следующий оператор и снова остановиться. Если оператор с точкой останова оказался вызовом функции, выполнить ее полностью и остановиться после возврата из нее.
- *Step Into* — выполнить только следующий оператор, но если он окажется вызовом функции, войти в нее и остановиться на первом же операторе внутри функции.
- *Step Out* — выполнить всю оставшуюся часть функции и остановиться на первом же операторе функции, которая вызвала данную функцию.
- *Run To Cursor* — продолжить выполнение программы и остановиться на операторе, на котором находится курсор.

Большая часть необходимой вам информации, имеющейся в распоряжении отладчика, оформлена в виде новых окон, которые рассматриваются в следующих разделах.

Команды и окна отладки

Visual Studio имеет мощный отладчик с богатым пользовательским интерфейсом. Он содержит команды меню, пиктограммы панели инструментов и окна, которые используются только при отладке.

Команды меню

Интерфейс пользователя, выполняющего отладку, начинается с команд некоторых меню, приведенных ниже и применяемых только для отладки. (Эти команды не рассматривались в приложении В.)

- Edit⇨Breakpoints
- View⇨Debug Windows⇨Watch
- View⇨Debug Windows⇨Call Stack
- View⇨Debug Windows⇨Memory
- View⇨Debug Windows⇨Variables
- View⇨Debug Windows⇨Registers
- View⇨Debug Windows⇨Disassembly
- Build⇨Start Debug⇨Go
- Build⇨Start Debug⇨Step Into
- Build⇨Start Debug⇨Run To Cursor
- Build⇨Start Debug⇨Attach to Process
- Build⇨Debugger Remote Connection

Конечно, для отладки можно применять и другие команды меню. Например, команду Edit⇨Go To можно использовать для перемещения курсора редактора к конкретной точке останова точно так же, как к строке, закладке или адресу.

Как только вы приступили к отладке, вместо меню Build появляется меню Debug, включающее следующие пункты.

- Debug⇨Go
- Debug⇨Restart
- Debug⇨Stop Debugging
- Debug⇨Break
- Debug⇨Step Into
- Debug⇨Step Over
- Debug⇨Step Out
- Debug⇨Run to Cursor
- Debug⇨Step Into Specific Funcion
- Debug⇨Exceptions
- Debug⇨Threads
- Debug⇨Show Next Statement
- Debug⇨Quick Watch

В меню Debug дублируются некоторые из пунктов каскадных меню Build⇨ Start Debug, но, кроме них, имеются и другие команды. Ннже обсуждаются отдельные команды меню.

Установка точек останова

Очевидно, что проще всего установить точку останова, поместив курсор на оператор программы, перед выполнением которого вы желаете остановиться. Собственно установка точки останова выполняется с помощью клавиши <F9> либо пиктограммы Insert/Remove breakpoint панели инструментов Build, которая имеет вид поднятой руки (ладони) (предполагается, что она будет ассоциироваться со знаком “Стоп”). Красная точка слева от оператора отметит точку останова, как показано на рис. Г.1.

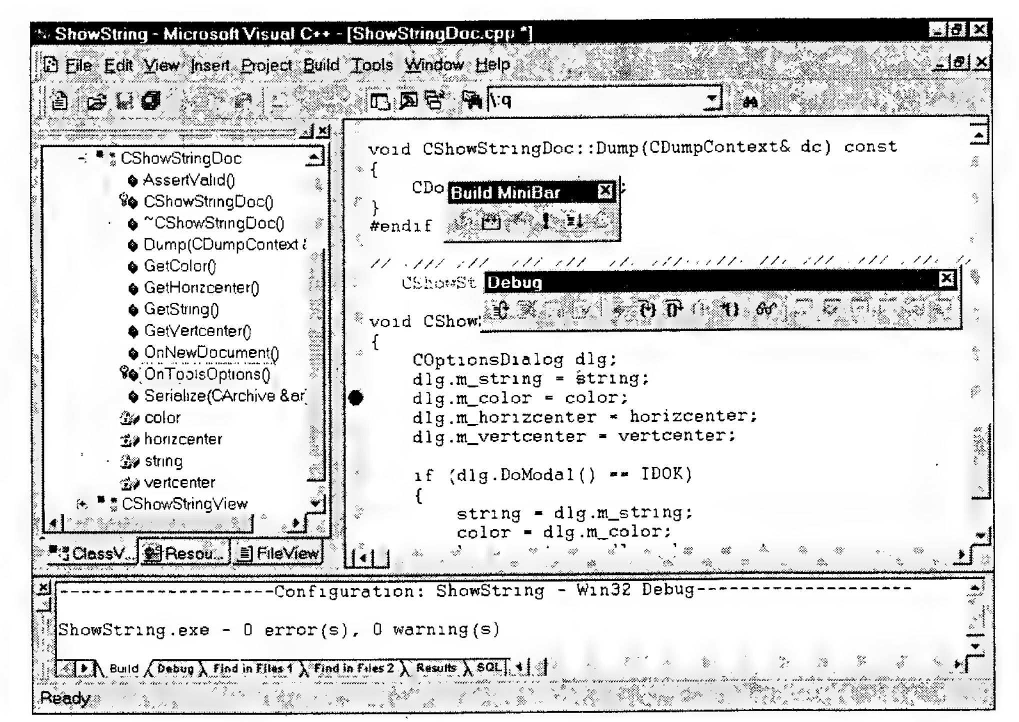


Рис. Г.1. Клавиша <F9> позволяет расположить точку останова на операторе в той строке, в которую помещен курсор

На заметку Приложение, которое отлаживается в этой главе, называется ShowString. Оно создано в главе 8.

При выборе пункта меню Edit⇨Breakpoints на экран выводится диалоговое окно для установки *простых* или *условных* точек останова. Например, можно остановиться там, где изменяется некоторое значение переменной. Процесс поиска в программе операторов, в которых изменяется значение переменной, и установка в них точек останова является утомительным занятием. Вместо этого лучше воспользоваться вкладкой Data диалогового окна Breakpoints, изображенного на рис. Г.2. При изменении значения переменной в окне сообщения можно узнать причину возникновения паузы в работе. После этого можно просмотреть текст программы и значения переменных.

Можно также установить условные точки останова, такие как *прервать выполнение программы, когда i достигает 100*, что при стократном выполнении тела цикла избавит вас от необходимости щелкать на Go 100 раз для того, чтобы просмотреть результаты.

Анализ значений переменных

При отладке программы после установки точки останова все происходит нормально до тех пор, пока не подойдет очередь выполнять оператор, на котором установлена точка останова. После этого на сцене появляется Visual Studio. Оно выводит поверх окна приложения несколько дополнительных окон и желтую стрелку возле красной точки, отмечающей текущую точку останова, как показано на рис. Г.3.

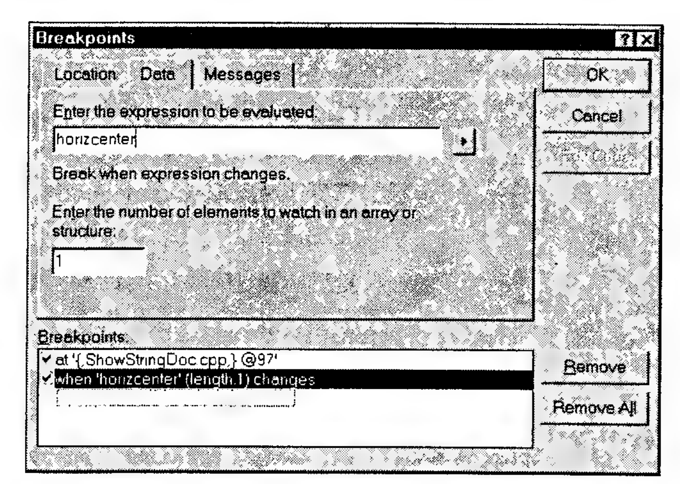


Рис. Г.2. Можно остановить выполнение программы, когда переменная или выражение изменяет свое значение

ShowString - Microsoft Visual C++ [break] - [ShowStringDoc.cpp]

File Edit View Insert Project Debug Tools Window Help

CShowStringDoc | (All class members) | OnTo… | Debug

```
void CShowStringDoc::Dump(CDumpContext
{
    CDocument::Dump(dc);
}
#endif //_DEBUG

// CShowStringDoc commands

void CShowStringDoc::OnToolsOptions()
{
    COptionsDialog dlg;
    dlg.m_string = string;
    dlg.m_color = color;
    dlg.m_horizcenter = horizcenter;
    dlg.m_vertcenter = vertcenter;

    if (dlg.DoModal() == IDOK)
```

Context: CShowStringDoc::OnToolsOptions()

Name	Value
color	0
dlg.m_color	-1

Auto | Locals | this

Name	Value

Watch1 | Watch2 | Watch3 | Watch4

Break at location breakpoint | Ln 97, Col 1

Рис. Г.3. Желтая стрелка показывает оператор программы, который должен быть выполнен следующим

Поместите указатель мыши на идентификатор переменной типа color или horizcenter. Появится окно Data Tip, в котором выведено текущее значение этой переменной. Можно просмотреть значения локальных переменных, затем продолжить выполнение программы и проверить их снова. Но существуют и другие способы доступа к значениям переменных.

Можно щелкнуть мышью на переменной и выбрать пункт меню Debug⇨QuickWatch или щелкнуть на пиктограмме QuickWatch панели инструментов (на ней изображены очки). Это приведет к выводу на экран окна QuickWatch, которое покажет вам значение переменной или выражения и позволит при желании добавить отслеживание этой переменной в окно Watch. Возможно, вас удивит применение такого способа, если можно с помощью окна Data Tip увидеть значение переменной без какого бы то ни было щелканья на кнопках. Дело в том, что окно Data Tip не поддерживает даже самых простых выражений, например dlg.m_horizcenter, а окно QuickWatch может это сделать, как видно на рис. Г.4. С его помощью можно также изменить "на ходу" значение переменной, чтобы исправить нелепые ошибки и увидеть результат.

На рис. Г.5 показан сеанс отладки после выполнения нескольких операторов программы после точки останова (вы скоро узнаете, как это сделать). Окна Watch и Variables раскрыты, чтобы яснее показать, что находится в каждом из них; в них добавлены два наблюдателя: один для переменной horizcenter, другой для переменной dlg.m_horizcenter. Как только пользователь щелкнет на OK диалогового окна Options, выполнение программы немедленно остановится. В данном случае пользователь во время сеанса работы в окне Options изменил строку, цвет и оба вида центрирования.

В окне Watch просто отображаются значения двух переменных, которые были добавлены в окно. Значение переменной horizcenter остается равным TRUE (1), так как оператор программы, который устанавливает значение этой переменной, еще не выполнялся. Значение переменной dlg.m_horizcenter равно FALSE, так как пользователь изменил состояние флажка, ассоциированного с переменной-членом. (Диалоговые окна, элементы управления и элементы управления, ассоциированные с переменными-членами, обсуждаются в главе 2.)

В окне Variables содержится много дополнительной информации, которая иногда затрудняет его использование. Локальная переменная dlg и указатель на объект, для которого эта функция-член активизирована, в окне Variables представлены в форме дерева: щелкнув на значке +, можно развернуть ветвь дерева, а после щелчка на значке – ветвь сворачивается. В дополнение к этой информации на экран выводится значение, возвращаемое функцией DoModal(), — (-1).

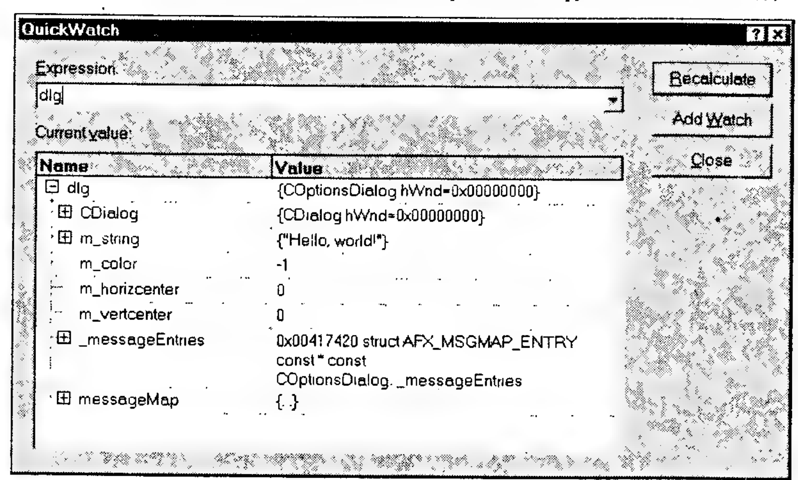


Рис. Г.4. С помощью диалогового окна QuickWatch вычисляются значения выражений. Их можно добавить в окно Watch, щелкнув на вкладке Add Watch

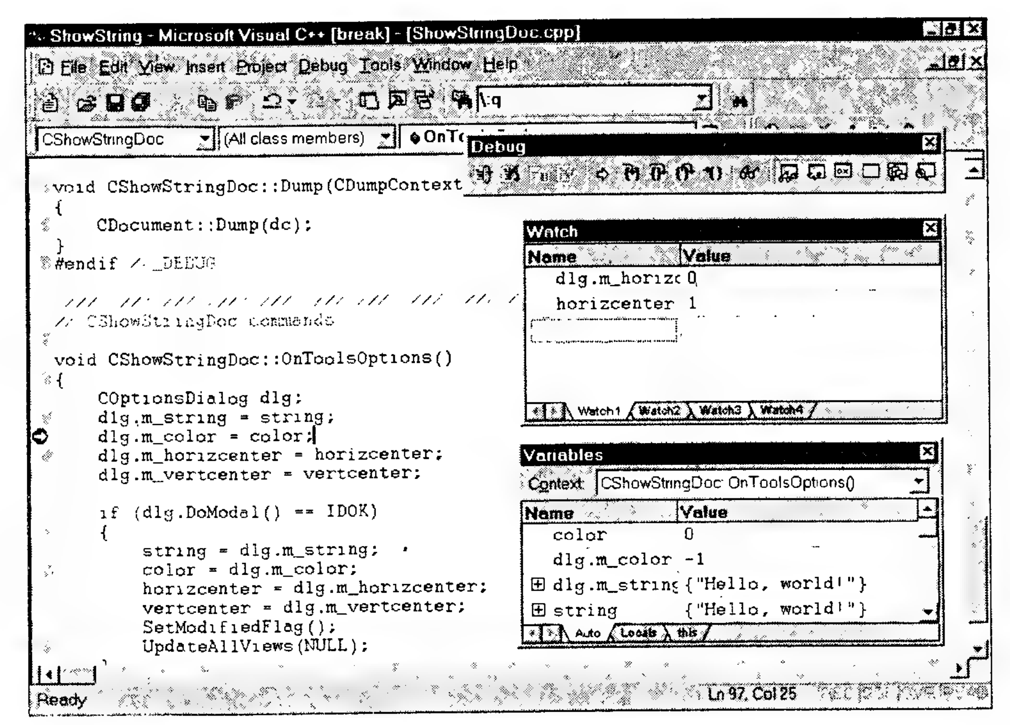


Рис. Г.5. Окна Watch *и* Variable *позволяют без труда просмотреть значения любых переменных*

Вверху окна Variables имеется раскрывающийся список Context. Если его раскрыть, то виден будет список имен ряда функций. Верхний элемент — это функция, в которой должен выполняться оператор (CShowStringDoc::OnToolsOptions()). Второй элемент — это функция, которая ее вызвала (DispatchCmdMsg()). Она распределяет командные сообщения. В главе 3 описываются команды и сообщения и обсуждается способ передачи управления в функцию обработки сообщений типа OnToolsOptions(). Здесь отладчик демонстрирует этот процесс буквально у вас на глазах. Дважды щелкнув на любом имени функции в списке, можно отобразить текст функции. При этом можно просмотреть переменные, локальные для функции, и другую информацию.

Окно Call Stack, показанное на рис. Г.6, немного проще для анализа, чем раскрывающийся список в окне Variables, хотя и показывает ту же информацию. Так же, как имена функций, можно просмотреть и параметры, переданные каждой функции. Обратите внимание на число 32771, возвращаемое большинством вызываемых функций. Если выбрать команды меню View⇨Resource Symbols, то можно заметить, что 32771 означает идентификатор ресурса ID_TOOLS_OPTIONS, ассоциированный с командами меню Tools⇨Options в окне ShowString (рис. Г.7).

```
Call Stack
CShowStringDoc::OnToolsOptions() line 104
_AfxDispatchCmdMsg(CCmdTarget * 0x004228e0 {CShowStringDoc}, unsigned i
CCmdTarget::OnCmdMsg(unsigned int 32771, int 0, void * 0x00000000, AFX_
CDocument::OnCmdMsg(unsigned int 32771, int 0, void * 0x00000000, AFX_C
CView::OnCmdMsg(unsigned int 32771, int 0, void * 0x00000000, AFX_CMDHA
CFrameWnd::OnCmdMsg(unsigned int 32771, int 0, void * 0x00000000, AFX_C
CWnd::OnCommand(unsigned int 32771, long 0) line 2088
CFrameWnd::OnCommand(unsigned int 32771, long 0) line 317
```

Рис. Г.6. Окно Call Stack *показывает, как вы сюда попали*

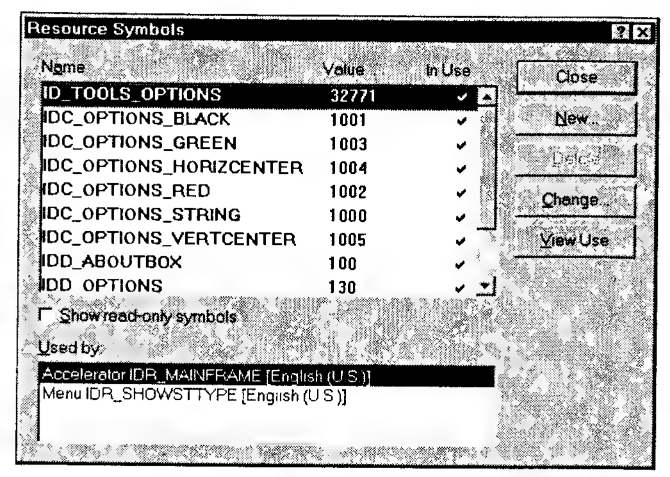


Рис. Г.7. Число 32771 относится к ID_TOOLS_OPTIONS

Пошаговое выполнение программы

Дважды щелкнув на имени функции в окне Call Stack или в списке Context окна Variables, нельзя запустить выполнение программы. Это просто даст возможность анализировать локальные переменные функций, которые вызывают функцию, выполняемую в момент останова. После просмотра всех желаемых переменных пора двигаться дальше. Несмотря на то что в меню Debug имеются команды Step Over, Step Into и т.д., большинство разработчиков используют пиктограммы панели инструментов или клавиши ускоренного вызова. Панель инструментов Debug можно увидеть на рис. Г.1 и Г.3. Задержите указатель мыши на каждой пиктограмме, чтобы увидеть, какие команды с ними связаны. Например, пиктограмма с изображением направленной вниз стрелки в фигурных скобках означает Step Into и соответствующая клавиша быстрого вызова — <F11>. При пошаговом выполнении программы желтая стрелка на левой границе окна текста программы движется, чтобы показать, какой оператор будет выполняться. Во время остановки выполнения программы можно добавить или убрать точки останова, проанализировать переменные или продолжить выполнение. В этом и заключается механизм отладки.

Редактирование и продолжение выполнения

Большинству разработчиков методика отладки хорошо знакома. Сначала строится проект, затем он запускается на выполнение и осуществляется проверка, не случилось ли чего, не запланированного при разработке. Далее выясняется причина неправильного поведения программы, редактируется код, вновь транслируется приложение и цикл повторяется до тех пор, пока все ошибки не будут устранены (по крайней мере, так кажется разработчику). Если вы имеете дело с большим проектом, то цикл "редактирование — повторная компиляция — поиск ошибки" может быть достаточно продолжительным. Особенно раздражает длительность повторной компиляции при малейшем изменении исходного кода программы. Не менее утомителен и повторный ввод данных в диалоговые окна приложения, если таковые существуют.

В версии Visual C++ 6.0 появилась возможность в ряде случаев избежать повторной трансляции и перезапуска приложения после редактирования фрагментов программного кода. Эта функция получила наименование *Edit and Continue* (Редактирование и продолжение выполнения).

Для активизации функции *Edit and Continue* начните с команды Tools⇨Options и щелкните на вкладке Debug. Установите флажок Debug commands invoke Edit and Continue,

как показано на рис. Г.8. Далее выберите команду Project⇨Settings и щелкните на вкладке C/C++. Настройте окно в левой части вкладки на установку опций отладки (в поле Settings for должно быть выбрано значение Win32 Debug). В поле списка Debug info должно быть выбрано значение Program Database for Edit and Continue (рис. Г.9). После того как окно Project Settings будет закрыто, скомпилируйте и скомпонуйте проект. Если пожелаете пользоваться новой функцией и при отладке других проектов, всегда начинайте работу с настройки указанных выше диалоговых окон.

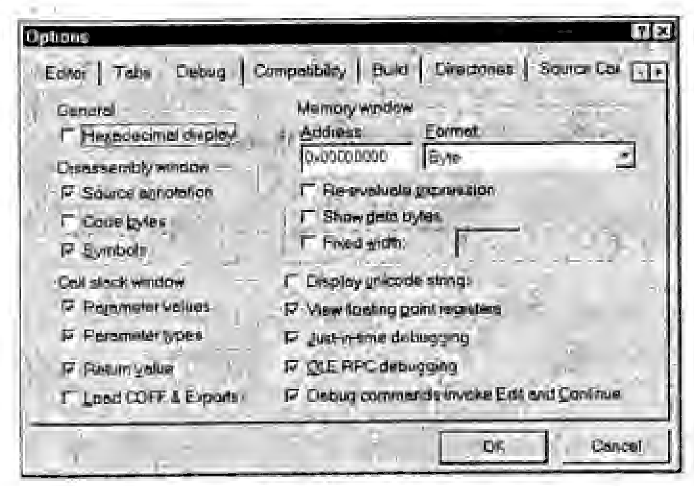


Рис. Г.8. Установите флажок Debug commands invoke Edit and Continue *на вкладке* Debug

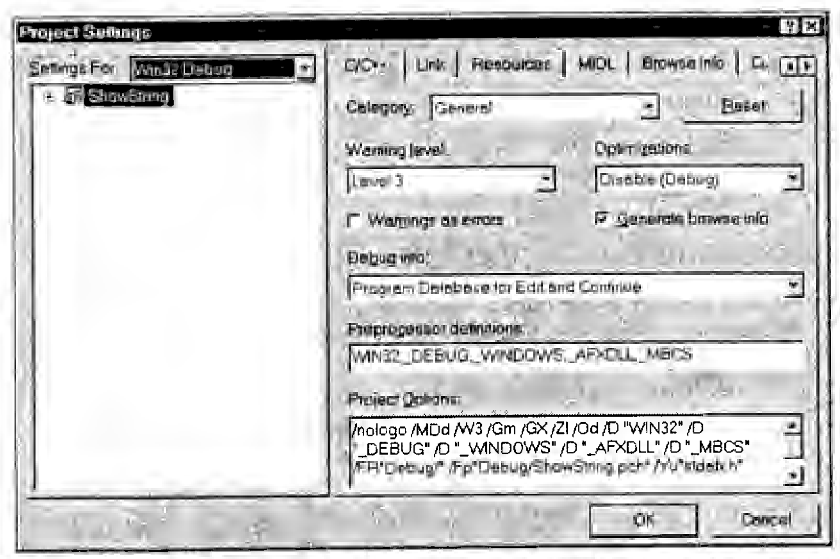


Рис. Г.9. Теперь в проекте будет формироваться информация для функции Edit and Continue

Теперь можно отлаживать приложение по обычной методике, но после каждого сеанса редактирования уже не нужно повторно компилировать проект. Можно просто выполнять следующий оператор программы. Если же система сочтет, что без повторной трансляции в данном случае не обойтись, она выведет строку в окно Output (вкладка Build) и сформирует привычное окно сообщения `One or More Files Are out of Date`. Вот теперь придется заново скомпилировать приложение.

Большинство простых изменений в тексте программы (наподобие корректировки логических условий в операторах `if` и `for` или установки других значений для переменных в операторах присваивания) не потребуют повторной компиляции. Среди более сложных корректировок, которые заставят систему потребовать повторной компиляции программы, нужно отметить следующие.

- Любые изменения в файлах заголовков, в том числе изменения кода встроенных (inline) функций
- Изменение определений классов C++
- Изменение прототипа любой функции
- Изменение кода глобальной функции, не являющейся членом какого-либо класса, или статического метода класса

Обратите внимание на одну особенность — после завершения сеанса отладки с использованием ***Edit and Continue*** обязательно выполните компиляцию откорректированного приложения. Дело в том, что внесенные вами изменения не попали в *выполняемый файл* приложения — они хранятся только в памяти и после закрытия проекта будут удалены.

Другие окна для отладки

До сих пор еще не обсуждались три окна, предназначенные для отладки: Memory, Registers и Disassembly. Эти окна обеспечивают отладку деталей программы, редко встречающихся в обычной процедуре отладки. С появлением каждой новой версии Visual C++ обстоятельства, при которых эти окна необходимы, встречаются все реже. Например, окно Registers используется для просмотра значения, только что возвращенного вызываемой функцией. Эта информация сейчас находится в окне Variables в более доступном формате.

Окно Memory

В окне, показанном на рис. Г.10, содержатся шестнадцатеричные значения в каждом байте памяти от 0x00000000 до 0xFFFFFFFF. Список этих значений очень длинный. А потому просматривать содержимое памяти с его помощью утомительно. Для ввода интересующего вас адреса проще использовать окно Address. Обычно эти адреса копируются (через буфер Clipboard, а не вручную) из окна Variables. Просмотр больших массивов или отслеживание тонкостей решения задач, зависящих от платформы *platform-dependent*, выполняется вручную.

```
Memory
Address: 0x00000000
00000000  ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ??????????
0000000A  ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ??????????
00000014  ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ??????????
0000001E  ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ??????????
00000028  ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ??????????
00000032  ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ??????????
0000003C  ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ??????????
```

Рис. Г.10. Вы можете анализировать распределение памяти, хотя необходимость в этом возникает очень редко

Окно Registers

При отладке на уровне ассемблера бывает полезно анализировать регистры. На рис. Г.11 показано окно Registers. Эти данные были взяты в той же точке выполнения программы, которая показана на рис. Г.5. Регистр EAX содержит значение 1, которое является значением, возвращаемым функцией DoModal().

Окно Disassembly

По умолчанию окно Disassembly заполняет весь экран, заменяя текст программы на C++ в основной рабочей зоне. В нем можно увидеть операторы ассемблера, сгенерированные для программы, написанной на C++, как показано на рис. Г.12. Изложение процесса отладки на уровне ассемблера выходит за рамки этой книги.

```
Registers
EAX = 00422934 EBX = 00008003
ECX = 004239F0 EDX = 00422934
ESI = 00141F10 EDI = 0012F900
EIP = 00402DAF ESP = 0012F840
EBP = 0012F90C EFL = 00000202
CS = 001B DS = 0023 ES = 0023
SS = 0023 FS = 0038 GS = 0000 OV=0
UP=0 EI=1 PL=0 ZR=0 AC=0 PE=0 CY=0

0012F8FC = 004228E0

ST0 = +0.000000000000000000e+0000
ST1 = +0.000000000000000000e+0000
ST2 = +0.000000000000000000e+0000
```

Рис. Г.11. Все регистры доступны для анализа или изменения

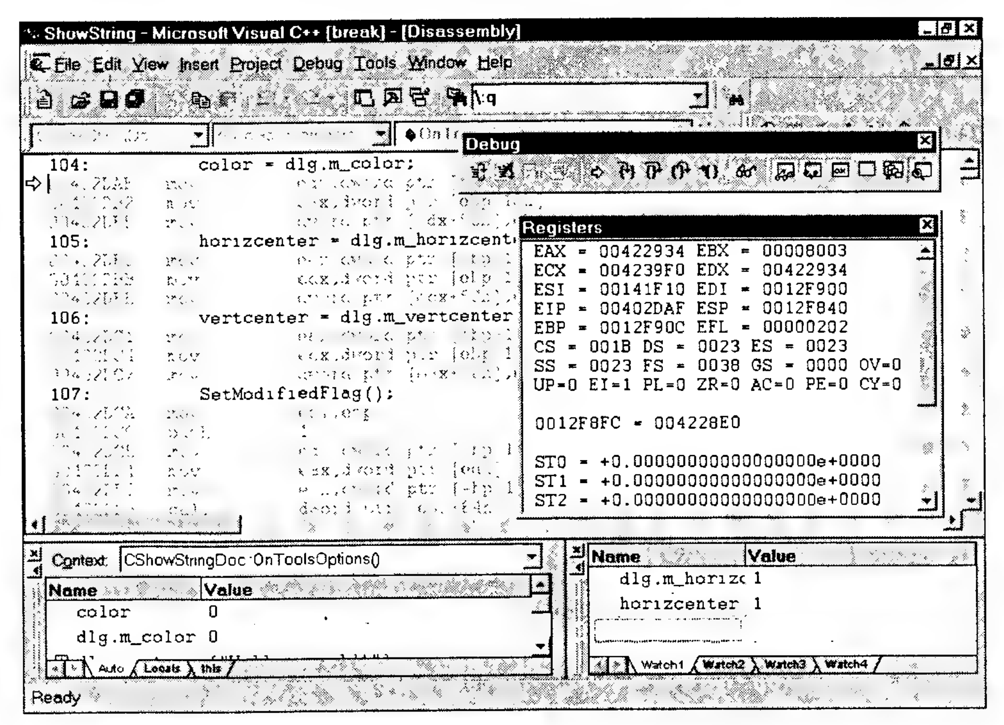


Рис. Г.12. Можно отлаживать сгенерированную ассемблерную программу

Применение утилиты MFC Tracer

Утилита MFC Tracer является автономным приложением с пунктом меню, интегрированным в Visual Studio. Для ее выполнения выберите команду Tools⇨MFC Tracer. На рис. Г.13 показано диалоговое окно Tracer.

Trace output will be available only while using the debugger.		OK
☑ Enable tracing	☐ Main message pump	Cancel
☐ OLE tracing	☐ Main message dispatch	
☐ Database tracing	☐ WM_COMMAND dispatch	
☐ Multiple application debugging	☐ Internet Client Tracing	

Рис. Г.13. Автономная утилита упрощает установку флагов трассировки

Трассировщик делает не очень много: он просто устанавливает флаги трассировки, которые управляют видом получаемого выхода при отладке. Попробуйте установить все флаги и выполнить ShowString, запустив его и затем закрыв. Сбросьте несколько флагов и посмотрите, как изменится полученный,результат.

Если все флаги установлены, ваше приложение будет выполняться медленно. Используйте трассировщик для установки только интересующих вас флагов. Это значительно проще, чем непрерывно менять переменную.

Метод Dump()

Метод Dump() существует во всех классах MFC. Если в программе что-то неправильно, код с дескриптором ошибки вызывает эту функцию, чтобы показать содержимое объекта. Методы Dump() можно разработать и для классов ваших объектов.

Классы MFC наследуют метод Dump()от класса CObject, где он определяется следующим образом:

```
virtual void Dump(CDumpContext& dc ) const;
```

Ключевое слово virtual предполагает, что вы должны заменить этот метод в производных классах, а const указывает на то, что функция Dump() не будет модифицировать состояние объекта.

Подобно операторам трассировки, метод Dump() в распространяемой (Release) версии программы исчезает. Это избавляет пользователей от необходимости просмотра ненужного вывода и делает версию короче. Вы должны реализовать условную компиляцию для любой написанной собственноручно функции Dump().

Объявление функции Dump()в файле заголовка выглядит так:

```
class CNewClass : public CObject
{ public
        //Другие члены.
        #ifdef _DEBUG
        virtual void Dump(CDumpContext& dc ) const;
        #endif
        //...
};
```

В файле реализации текст программы, который содержит тело функции, может выглядеть следующим образом:

```
#include "cnewclass.h"
#ifdef _DEBUG
void CNewClass::Dump(CDumpContext& dc ) const
{
        CObject:: Dump( dc ); //Член класса родителя;
        //возможно, индивидуальные элементы функции работают, как cout.
        dc << "member: " << /*здесь члены класса */ endl;
}
 #endif
```

Видно, что текст программы функции Dump() очень похож на программу стандартного вывода с объектом класса cout или преобразования в последовательную форму для архива. В вашем распоряжении имеется объект класса CDumpContext, названный dc, и можно передать в этот объект текст или значение с помощью оператора <<. Если вы с ним не знакомы, прочтите главу 7.

Пример использования классов CDumpContext, CFile и объекта afxDump

В примере приложения используется отладочный класс MFC CDumpContext и глобальный объект afxDump. Программа вывода в окно отладки и в объект класса CFile приведена в листинге Г.1. Для самостоятельного выполнения этого примера создайте консольное приложение, как описано в главе 28, и пустой файл программы на C++ Dump.cpp. Введите в него этот текст программы, постройте отладочную версию проекта и выполните его.



Проект Dump находится в папке RefC компакт-диска, прилагаемого к этой книге.

Листинг Г.1. Демонстрация отладочного класса MFC CDumpContext и CFile файлом Dump.Cpp

```
#include <afx.h>
// _DEBUG определен при построении отладочной версии.
class CPeople : public CObject
{
public:
        // Конструктор.
        CPeople( const char * name );
        // Деструктор.
                virtual ~CPeople();
                #ifdef _DEBUG
                virtual void Dump(CDumpContext& dc) const;
                #endif
private:
                CString * person;
};
        // Конструктор.
                CPeople::CPeople( const char * name) : person( new CString(name));
        // Деструктор.
                CPeople::~CPeople(){ delete person; }
#ifdef _DEBUG
        void CPeople::Dump( CDumpContext& dc ) const
        {
```

```
          CObject::Dump(dc);
                dc << person->GetBuffer( person->GetLength() + 1);
        }
#endif
int main()
{
        CPeople person1("Kate Gregory");
        CPeople person2("Clayton Walnum");
        CPeople person3("Paul Kimmel");
   // Использует существующий afxDump с виртуальной
   // функцией-членом Dump().
        person1.Dump( afxDump );
        // Объект класса CFile.
        CFile dumpFile("dumpout.txt", CFile::modeCreate |
        CFile::modeWrite);
        if( !dumpFile )
        {
        afxDump << "File open failed.";
        }
        else
        {
        // Dump с другими функциями CDumpContext.
        CDumpContext context(&dumpFile);
        person2.Dump(context);
    }
        return 0;
}
```

Этот файл содержит определение класса, все тексты функций-членов класса и функцию main() для ее выполнения как консольного приложения. Каждая часть этого файла обсуждается ниже. Класс — это простенький упаковщик, работающий с указателем на класс CString, который создает объект класса CString в конструкторе с помощью оператора new и удаляет его в деструкторе. Очевидна бесполезность применения этого класса для чего-нибудь другого, кроме демонстрации функции Dump().

Файл начинается с директивы включения файла заголовка <afx.h>, который содержит определение класса CObject и обеспечивает доступ к afxDump. Следующий фрагмент в этом файле определяет класс CPeople, производный от класса CObject. Обратите внимание на замещение виртуального метода Dump() и директивы условной трансляции. (Любые вызовы функции Dump() должны быть организованы в программе точно так же.)

После конструктора и деструктора стоит текст для функции класса CPeople::Dump(). Заметьте, как он обрамлен директивами условной трансляции. Вызов метода CObject::Dump() использует результаты работы программистов в MFC, выводя информацию, содержащуюся во всех объектах.

Наконец, функция main() тестирует этот маленький класс. Она создает три экземпляра класса CPeople и выводит данные первого из них.

Для второго экземпляра класса CPeople эта программа создает и открывает объект класса CFile, передавая текстовую строку конструктору. Если файл открылся, она создает экземпляр класса CDumpContext из файла и передает этот контекст в Dump(), а не, как обычно, в afxDump().

Результат выполнения этой программы должен походить на то, что показано на рис. Г.14. Файл document.txt будет содержать такие строки:

```
a CObject at $76FDE4
Kate Gregory
```

Первая строка, как для окна отладки, так и для файла, пришла из класса CObject:: Dump() и предоставляет информацию о типе объекта и его адресе. Вторая строка появилась из рас-

ширенной вами программы и является просто содержимым переменной Cstring — членом экземпляра класса CPeople.

Если при компоновке версии Debug этого продукта получено сообщение об ошибке, относящейся к _beginthreadex и _endthreadex, необходимо изменить некоторые установки. По умолчанию консольные приложения односвязные, но MFC является многосвязной средой. Если включить в приложение файл afx.h и поместить его в MFC, то это приложение становится само по себе несовместимым с односвязностью по умолчанию. Чтобы разрешить проблему, выберите Project Settings и щелкните на вкладке C/C++. Из раскрывающегося списка в верхней части диалогового окна выберите Code Generation. В списке Use run-time library выберите Debug Multithreaded. (Итоговое диалоговое окно приведено на рис. Г.15.) Щелкните на кнопке OK и перестройте проект. Обычно необходимо изменять установки и для версии Release, но, так как вызовы функции Dump() ограничены директивами анализа _DEBUG, эта программа, в принципе, не будет компилировать версию Release.

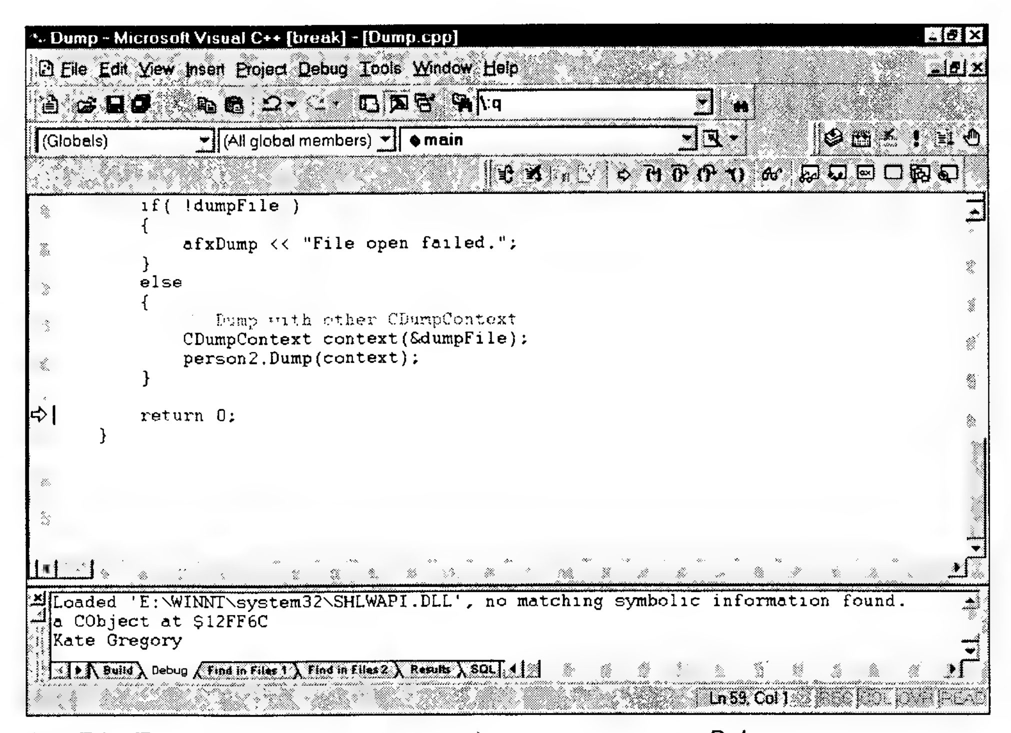


Рис. Г.14. При использовании afxDump вывод направляется в окно Debug

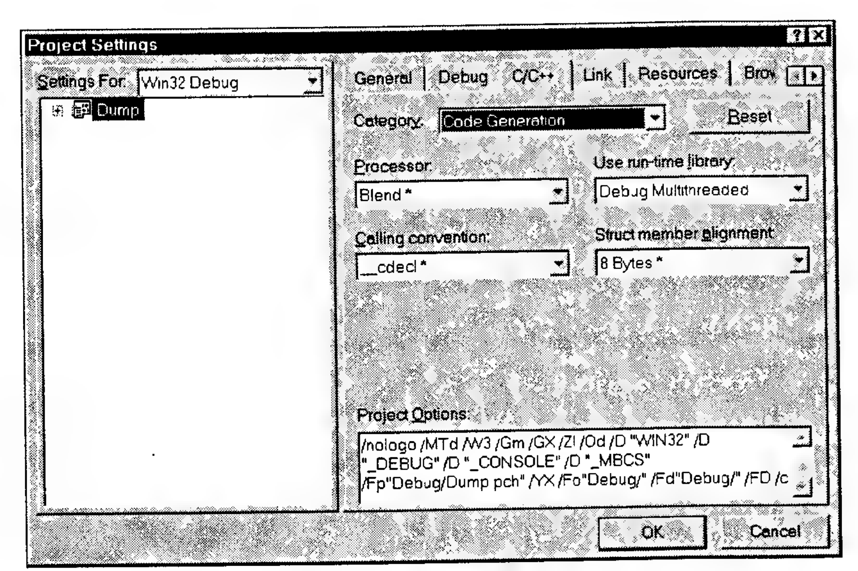


Рис. Г.15. Для использования MFC в консольном приложении выберите в списке **Use Runtime Library** *элемент* `Debug Multithreaded`

Теперь, когда вы увидели базовые средства отладки в действии, можете включать их в собственные приложения. Вы быстро обнаружите ошибки, поймете, как работают другие аналогичные программы, увидите маршруты движения сообщений и другие закулисные чудеса. Если вы находите удовольствие в отладке, не волнуйтесь: об этом никто не должен знать!



Макросы и глобальные объекты MFC

В этом приложении...

Десять категорий макросов и глобальных переменных

Информация о приложениях и административные функции

Разделители комментариев ClassWizard

Набор классов для работы с популярными структурами данных

Форматирование класса CString и вывод на экран окна сообщения

Типы данных

Использование сервиса диагностики

Обработка исключений

Использование макросов карты сообщения

Сервис модели объектов во время выполнения

Стандартные идентификаторы команд и окон

При написании программ многие типы данных и операций используются многократно. В одних случаях вам необходимо сделать что-нибудь простое, например создать переносимый тип данных *целый*, в других — создать что-либо более сложное, например извлечь слово из длинного слова или запомнить положение указателя мыши. Как вам может быть известно, при компилировании программы с помощью Visual C++ используется много определенных ранее констант и переменных. Применяя их, можно сэкономить время, затраченное на написание программы. Кроме того, программы становятся более компактными и легче читаются другими программистами. В таблицах, представленных ниже, приведены наиболее важные глобально доступные переменные, константы и макросы.

Десять категорий макросов и глобальных переменных

Разделение множества констант, макросов и глобальных переменных на категории облегчает оперирование ими. Список этих категорий представлен ниже. В следующих разделах описывается каждая из категорий и символы, которые они определяют.

- Информация о приложении и администрирование
- Разделители комментариев ClassWizard
- Коллекция класса справочников
- Форматирование класса CString и отображение окна сообщения
- Типы данных
- Сервис диагностики
- Обработка исключительных ситуаций
- Карты сообщений
- Сервисные модели объекта во время выполнения
- Стандартные идентификаторы команд и окон

Информация о приложениях и административные функции

В обычном приложении Visual C++ содержится только один объект приложения и много объектов, созданных из других классов MFC. В связи с этим часто возникает необходимость в получении информации о приложении в различных частях программы. Visual C++ определяет набор глобальных функций, возвращающих данную информацию в любой класс программы. Эти функции, перечисленные в табл. Д.1, могут быть вызваны из любой части программы MFC. Например, вам срочно потребовалось считать указатель на основное окно приложения. Эту задачу решает вызов следующей функции:

```
CWnd* pWnd = AfxGetMainWnd();
```

Таблица Д.1. Информация о приложениях и администрирование

Функция	Назначение
AfxBeginThread()	Создает новый поток (см. главу 27)
AfxEndThread()	Завершает параллельное выполнение функций
AfxGetApp()	Считывает указатель приложения — экземпляр класса CWinApp
AfxGetAppName()	Считывает имя приложения
AfxGetInstanceHandle()	Считывает дескриптор экземпляра приложения
AfxGetMainWnd()	Считывает указатель на основное окно приложения
AfxGetResourceHandle()	Считывает дескриптор ресурса приложения
AfxGetThread()	Считывает указатель на объект класса CWinThread
AfxRegisterClass()	Регистрирует оконный класс в MFC DLL
AfxRegisterWndClass()	Регистрирует оконный класс Windows в приложении MFC
AfxSetResourceHandle()	Устанавливает дескриптор экземпляра, который определяет по умолчанию путь загрузки ресурса приложения
AfxSocketInit()	Инициализирует интерфейс Windows Socets (см. главу 18)

Разделители комментариев ClassWizard

В Visual С++ определен ряд разделителей, используемых ClassWizard для поддержки трассировки, а также для локализации особых областей программы (табл. Д.2). И хотя эти макросы используются разработчиками очень редко, они встроены в приложения AppWizard, так что желательно иметь представление о том, что же они делают.

Таблица Д.2. Разделители ClassWizard

Разделитель	Назначение
AFX_DATA	Начинает и заканчивает объявление переменных-членов в файлах заголовка, имеющих отношение к диалоговому обмену данными
AFX_DATA_INIT	Начинает и заканчивает диалоговый обмен данными при инициализации переменных в конструкторе класса диалогового окна
AFX_DATA_MAP	Начинает и заканчивает последовательность вызова функций обмена данными в функции-члене DoDataExchange() класса диалогового окна
AFX_DISP	Начинает и заканчивает объявление автомата в файлах заголовка
AFX_DISP_MAP	Начинает и завершает секцию автомата в файлах реализации
AFX_EVENT	Начинает и заканчивает объявление события ActiveX в файлах заголовка
AFX_EVENT_MAP	Начинает и завершает секцию событий ActiveX в файлах реализации
AFX_FIELD	Начинает и заканчивает объявление переменных-членов в файлах заголовка, ассоциированных с передачей полей записи базы данных
AFX_FIELD_INIT	Начинает и заканчивает инициализацию переменных-членов, связанных с передачей полей записи в конструкторах классов
AFX_FIELD_MAP	Начинает и заканчивает последовательность вызовов функций передачи полей записи в функциях DoFieldExchange() — членах классов записей
AFX_MSG	Начинает и заканчивает последовательность элементов карты сообщений, которую ClassWizard вставляет в файлы заголовка
AFX_MSG_MAP	Начинает и заканчивает последовательность элементов карты сообщений
AFX_VIRTUAL	Начинает и заканчивает последовательность замещения виртуальных функций в файлах заголовка

Набор классов для работы с популярными структурами данных

Поскольку определенные типы структур данных часто используются в программировании, в MFC определены классы коллекций, которые позволяют быстро получать доступ к инициализированным структурам данных общего назначения и упрощают операции с ними. MFC содержит классы коллекций для массивов, связных списков и таблиц соответствий. Каждый из этих типов коллекций содержит элементы, которые представляют отдельные части данных, составляющих коллекцию. Чтобы было легче получить доступ к этим элементам, MFC определяет набор функций, созданных из шаблонов (см. главу 26). Функции приведены в табл. Д.3. С их помощью можно запрограммировать реализацию для каждого конкретного типа данных.

Например, если вам нужен отсортированный список, функция, которая вставляет новые элементы в список, должна уметь сравнивать два объекта класса `Truck` или два объекта класса `Employee`, чтобы решить, куда поместить новый `Truck` или `Employee`. Вы реализуете функцию `CompareElements()`для класса `Truck` или `Employee`, а затем используете ее для принятия решения, куда поместить новые дополнения к коллекции.

Таблица Д.3. Методы классов для работы с популярными структурами данных

Метод	Назначение
`CompareElements()`	Проверяет элементы на равенство
`ConstructElements()`	Конструирует новые элементы (работает аналогично конструктору класса)
`DestructElements()`	Разрушает элементы (работает аналогично деструктору класса)
`DumpElements()`	Обеспечивает диагностический вывод в текстовой форме
`HashKey()`	Вычисляет ключи хэширования
`SerializeElements()`	Записывает элементы в архив или восстанавливает элементы из архива

Форматирование класса `CString` и вывод на экран окна сообщения

Если вы много программировали в среде Visual C++, то вам известно, что MFC содержит специальный класс `CString`, который делает обработку текстовых данных в программе на C++ менее громоздкой. Объекты класса `CString` широко используются в программах MFC и описываются в приложении Е. Иногда использование класса `CString` является не лучшим решением, например в случае элементов таблиц строк. Глобальные функции, которые замещают символы управления форматом в таблицах строк, обеспечиваются функцией `Format()` класса `CString`. Кроме того, имеется глобальная функция для отображения окна сообщения (табл. Д.4).

Таблица Д.4. Функции форматирования класса `CString` и вывода окна сообщения

Функция	Назначение
`AfxFormatString1()`	Замещает символы управления форматом (такие как `%1`) в строке ресурса данной строкой
`AfxFormatString2()`	Замещает символы управления форматом `%1` и `%2` в строке ресурса данными строками
`AfxMessageBox()`	Отображает окно сообщения

Типы данных

Наиболее часто используются константы, которые определяют переносимый набор типов данных. Вы встречались с этими константами много раз. Для их идентификаторов использованы прописные буквы. С одними из них можно было познакомиться еще в Windows SDK. Другие же включены только как часть Visual C++. Эти константы используются так же, как любые другие типы данных. Например, для объявления булевой переменной можно написать:

```
BOOL flag;
```

В табл. Д.5 приведены чаще всего используемые типы данных, определяемые Visual C++ для Windows 95/98 и NT. Поиск в справке тематического индекса любого из этих типов приведет вас к странице в интерактивной справке, в которой приведен список всех типов данных, используемых в MFC и Windows SDK.

Таблица Д.5. Типы данных общего назначения

Обозначение	Тип данных
BOOL	Булево значение
BSTR	32-битовый указатель
BYTE	8-битовое целое без знака
COLORREF	32-битовое значение цвета
DWORD	32-битовое целое без знака
LONG	32-битовое целое со знаком
LPARAM	32-битовый параметр оконной процедуры
LPCRECT	32-битовый константный указатель в структуре RECT
LPCSTR	32-битовый указатель строки-константы
LPSTR	32-битовый указатель строки
LPVOID	32-битовый указатель на void
LRESULT	32-битовое значение, возвращаемое оконной процедурой
POSITION	Позиция элемента в коллекции
UINT	32-битовое целое без знака
WNDPROC	32-битовый указатель оконной процедуры
WORD	16-битовое целое без знака
WPARAM	32-битовый параметр оконной процедуры

Использование сервиса диагностики

Подготовка текста программы — это только начало тернистого пути создания работающего приложения. Наступает трудное время анализа всех ветвей программы, работы с отладчиком и выявления всех глупостей, скрытых в вашей программе. К счастью, в Visual C++ предусмотрено много макросов, функций и глобальных переменных, которые можно использовать для включения диагностических функций в проект. Используя эти средства, можно вывести некоторые данные в окно отладки, проверить целостность блоков памяти и многое другое. В табл. Д.6 приведены эти полезные диагностические макросы, функции и глобальные переменные.

Таблица Д.6. Диагностические макросы, функции и глобальные переменные

Символ	Назначение
AfxCHeckMemory()	Проверяет целостность выделенной памяти
AfxDoForAllClasses()	Вызывает данную итерационную функцию для всех классов, порожденных классом CObject и включающих анализ типа во время выполнения
AfxDoForAllObjects()	Вызывает данную итерационную функцию для всех объектов классов, производных от класса CObject, которые созданы с помощью оператора new
afxDump	Глобальный объект класса CDumpContext, который позволяет программе посылать информацию в окно отладки
AfxDump()	Выводит состояние объекта во время отладки
AfxEnableMemoryTracking()	Переключает трассировку памяти
AfxIsMemoryBlock()	Проверяет, было ли успешным выделение памяти
AfxIsValidAddress()	Проверяет корректность диапазона изменения адресов памяти для программы
AfxIsValidString()	Анализирует корректность указателя на строку
afxMemDF	Глобальная переменная, которая управляет диагностикой выделения памяти. Допустимые значения: allocMemDF, DelayFreeMemDF, checkAlwaysMemDF
AfxSetAllocHook()	Устанааливает функцию Hook(), определенную пользователем, которая вызывается при выделении памяти
afxTraceEnabled	Глобальная переменная, которая разрешает или запрещает вывод TRACE
afxTraceFlags	Глобальная переменная, которая активизирует функции сообщения MFC
ASSERT	Печатает сообщение и прекращает выполнение программы, если соответствующее выражение равно FALSE (см. главу 24)
ASSERT_VALID	Обеспечивает доступность объекта путем вызова функции AssertValid() класса объекта
DEBUG_NEW	Используется в месте расположения оператора new для трассировки проблем утечки памяти (см. главу 24)
TRACE	Создает форматированную строку для отладочного вывода (см. главу 24)
TRACE0	То же, что и TRACE, но не требует аргументов в строке формата
TRACE1	То же, что и TRACE, но требует один аргумент в строке формата
TRACE2	То же, что и TRACE, но требует два аргумента в строке формата
TRACE3	То же, что и TRACE, но требует три аргумента в строке формата
VERIFY	Похожа на ASSERT, но VERIFY вычисляет выражение в обеих версиях программы — отладочной (Debug) и распространяемой (Release). Если проверка не удалась, печатается сообщение, но программа останавливается только в версии Debug

Обработка исключений

Одним из самых свежих элементов языка С++ является элемент *исключение*, дающий программе более широкие возможности управления дескрипторами ошибок (см. главу 26). До его появления в языке разработчики использовали для этих же целей макросы. Сейчас исключения являются неотъемлемыми элементами языка Visual С++. Набор функций упрощает оперирование исключениями различных типов. Эти макросы и функции приведены в табл. Д.7.

Таблица Д.7. Особые макросы и функции

Символ	Назначение
AfxAbort()	Прекращает выполнение приложения на фатальной ошибке
AfxThrowArchiveException()	Запускает исключение архива
AfxThrowDAOException()	Запускает CDaoExeption
AfxThrowDBException()	Запускает CDBExeption
AfxThrowFileException()	Запускает исключение файла
AfxThrowMemoryException()	Запускает исключение памяти
AfxThrowNotSupportedException()	Запускает неподдерживаемое исключение
AfxThrowOleDispatchException()	Запускает исключение OLE-автомата
AfxThrowOleException()	Запускает исключение OLE
AfxThrowResourceException()	Запускает исключение "ресурс не найден"
AfxThrowUserException()	Запускает исключение конечного пользователя
AND_CATCH	Начинает блок программы, который захватывает определенное исключение, не перехваченное в выполняемом блоке
AND_CATCH_ALL	Начинает блок программы, которая захватит все исключения, не захваченные в выполняемом блоке TRY
CATCH	Начинает блок программы для захвата исключения
CATCH_ALL	Начинает блок программы для захвата всех исключений
END_CATCH	Заканчивает блоки программы CATCH или CATCH_ALL
END_CATCH_ALL	Заканчивает блок программы CATCH_ALL
THROW	Запускает данное исключение
THROW_LAST	Запускает последнее исключение в следующий обработчик
TRY	Начинает блок программы, которая включает обработку исключения

Использование макросов карты сообщения

Windows — это операционная система, управляемая событиями. Это означает, что любое приложение Windows должно обрабатывать множество сообщений между приложением и системой. MFC "покончила" с использованием операторов switch, которые программисты в среде Windows должны были включать в программу для обработки сообщений, и заменила их картой сообщений. *Карта сообщений* — это не что иное как таблица, отождествляющая сообщение с программой его обработки (см. главу 3). Для упрощения объявления и заполнения этих таблиц данными в Visual C++ предусмотрен набор макросов карт сообщений. Миогие из этих макросов, собранных в табл. Д.8, должны быть уже знакомы программистам, имеющим опыт работы с MFC.

Таблица Д.8. Макросы карт сообщений

Макрос	Назначение
BEGIN_MESSAGE_MAP	Начинает определение карты сообщения
DECLARE_MESSAGE_MAP	Начинает объявление карты сообщения
END_MESSAGE_MAP	Заканчивает определение карты сообщения
ON_COMMAND	Начинает элемент карты сообщения
ON_COMMAND_RANGE	Начинает элемент карты сообщения, который отображает соответствие множества сообщений одной функции
ON_CONTROL	Начинает элемент, связанный с элементом управления в карте сообщения
ON_CONTROL_RANGE	Начинает элемент карты сообщения, в котором множество элементов управления соответствует одной функции обработки
ON_MESSAGE	Начинает элемент сообщения пользователя в карте сообщения
ON_REGISTERED_MESSAGE	Начинает элемент зарегистрированного сообщения пользователя в карте сообщения
ON_UPDATE_COMMAND_UI	Начинает элемент объявления команды в карте сообщения
ON_UPDATE_COMMAND_UI_RANGE	Начинает элемент объявления команды в карте сообщения, в котором множеству сообщений обновления соответствует одна функция обработки

Сервис модели объектов во время выполнения

Часто при использовании программ возникает необходимость получить информацию о выполняемых классах. В MFC имеется макрос для получения такого типа информации в структуре CRuntimeClass. Кроме того, интегрированная среда MFC-приложения опирается на набор макросов для объявления и определения возможностей исполняющей системы (такие как вывод/загрузка объектов и создание динамического объекта). При использовании AppWizard вам встречались эти макросы в сгенерированных программных файлах. Опытные программисты могут использовать их в собственных разработках. В табл. Д.9 приведены динамические макросы и их назначение.

Таблица Д.9. Сервисные динамические макросы

Символ	Назначение
DECLARE_DYNAMIC	Используется в объявлениях класса для разрешения доступа к информации о классе во время выполнения
DECLARE_DYNCREATE	Используется в объявлениях класса, чтобы разрешить создание динамического класса (порожденного из CObject). Разрешает также допуск к информации об этом классе во время выполнения
DECLARE_OLECREATE	Используется в объявлениях класса, чтобы разрешить создание объекта с OLE-автоматом
DECLARE_SERIAL	Используется в объявлениях класса, чтобы разрешить вывод/загрузку данных объекта. Разрешает допуск к информации о классе во время выполнения
IMPLEMENT_DYNAMIC	Используется в реализации класса, чтобы разрешить допуск к информации о классе во время выполнения

Символ	Назначение
IMPLEMENT_DYNCREATE	Используется в реализации класса, чтобы разрешить создание динамического объекта. Разрешает также допуск к информации о классе во время выполнения
IMPLEMENT_OLECREATE	Используется в реализации класса, чтобы разрешить создание объекта с OLE
IMPLEMENT_SERIAL	Используется в реализации класса, чтобы разрешить вывод/загрузку данных объекта. Разрешает допуск к информации о классе во время выполнения
RUNTIME_CLASS	Возвращает структуру CRuntimeClass для данного класса

Стандартные идентификаторы команд и окон

Существует множество стандартных сообщений, которые могут быть выработаны пользователем Windows-приложений. Например, программа присылает сообщение, когда пользователь выбирает команду стандартного меню File или Edit. Каждая из этих стандартных команд представлена идентификатором. Для того чтобы программисту было легче определять дюжины идентификаторов, часто используемых в приложениях Windows, Visual C++ определяет эти символы в файле AFXRES.H. Одни идентификаторы имеют очевидное назначение (например, ID_FILE_OPEN), а другие используются внутри MFC для решения широкого круга задач — от установки соответствия между стандартными сообщениями Windows и функциями их обработки до определения идентификаторов таблиц строк и идентификаторов стилей панелей инструментов и строк состояния.

Этих идентификаторов слишком много, чтобы можно было привести здесь их список. Однако, если у вас есть желание познакомиться с ними, загрузите файл AFXRES.H из папки установки продукта Visual C++ на своем компьютере.

ПРИЛОЖЕНИЕ E

Полезные классы

В этом приложении...

Классы массивов

Классы списков

Классы ассоциированных списков

Шаблоны класса коллекций

Класс CString

Классы для работы со временем

Библиотека MFC включает не только классы для программирования графического интерфейса Windows. Она также предоставляет в распоряжение программиста множество классов, которые могут быть использованы для обработки таких специфических структур данных, как списки, массивы, переменные времени и даты, коллекции. Освоив работу с ними, вы сможете значительно повысить качество своих программ и упростить программирование сложных процедур обработки данных.

Например, классы MFC, которые оперируют массивами, допускают динамическое изменение размерности массива. В результате вам не придется создавать массив с запасом по размерности, который будет занимать огромный объем памяти. А экономное расходование такого ресурса, как память, — это большое преимущество профессионально разработанной программы. Похожими достоинствами обладают и другие классы MFC, оперирующие коллекциями.

Классы массивов

Классы массивов из состава библиотеки MFC позволяют создать объект, представляющий собой одномерный массив из экземпляров любого класса. Эти объекты массивов функционируют практически так же, как и всем знакомые массивы простых переменных, но MFC обеспечивает расширение или сжатие массива по ходу выполнения программы. А это, в свою очередь, означает, что впредь вам не придется заранее беспокоиться об объявлении размерности массива в расчете на самый невероятный случай. В результате навсегда уйдут в прошлое времена, когда массивы нерационально отнимали память в расчете на сочетание условий выполнения программы, которое бывает раз в сто лет. Ваше приложение будет отбирать у операционной системы столько памяти, сколько ему действительно сейчас необходимо.

В состав классов MFC, поддерживающих работу с массивами, входят `CByteArray`, `CDWordArray`, `CObArray`, `CPtrArray`, `CUIntArray`, `CWordArray` и `CStringArray`. Как понятно из названий классов, каждый из них предназначен для хранения данных определенного типа. Например, `CUIntArray` — это класс массива, хранящего целые без знака (такого типа класс будет использован в примере, который рассматривается в этом разделе). С другой стороны, класс `CPtrArray` используется для хранения указателей на тип `void`, а `CObArray` — массив объектов. Все классы массивов практически идентичны за исключением того, что они содержат элементы разных типов. Таким образом, освоив работу с одним классом, можете считать, что вы постигли таинства всех. В табл. E.1 представлены методы классов массивов и приведено их описание.

Таблица E.1. Методы классов массивов

Функция	Назначение
`Add()`	Добавляет элемент в конец массива и при необходимости увеличивает его размер
`ElementAt()`	Получает ссылку на указатель элемента массива
`FreeExtra()`	Освобождает неиспользуемую дополнительную память
`GetAt()`	Считывает элемент по заданному индексу
`GetSize()`	Считывает количество элементов в массиве
`GetUpperBound()`	Считывает значение верхней границы (upper bound) массива — наименьшего индекса элемента массива
`InsertAt()`	Вставляет элемент в массив по заданному индексу, причем существующие элементы сдвигаются вверх (т.е. в сторону возрастания индекса)
`RemoveAll()`	Удаляет все элементы массива
`RemoveAt()`	Удаляет элемент массива по заданному индексу

Функция	Назначение
SetAt()	Устанавливает значение элемента массива по заданному индексу. Поскольку эта функция не изменяет размерности массива, значение индекса должно находиться в допустимых пределах
SetAtGrow()	Устанавливает значение заданного элемента, причем при необходимости размер массива увеличивается. Это происходит в случае, если заданное значение индекса превышает существующий в момент обращения размер массива
SetSize()	Устанавливает исходный размер массива и параметр приращения размера массива. Если при увеличении размера массива будет отбираться память сразу для нескольких элементов (с запасом), это сэкономит в дальнейшем время (для некоторых элементов "тепленькое" местечко в массиве уже подготовлено), но несколько увеличит расход памяти (ресурса операционной системы). Ну что ж, за все в этом мире приходится платить

Шаблоны массивов

Поскольку все упомянутые выше классы массивов различаются только классом (типом) элементов, которые содержатся в массиве, возникает совершенно естественное желание использовать шаблоны языка C++. И действительно, эти классы фактически предвосхитили появление в Visual C++ шаблонов. В библиотеке Standard Template Library, о которой шла речь в главе 26, существует шаблон вектора, который может хранить элементы любого простого типа. Среди разработчиков распространено мнение, что использование классов массивов MFC значительно проще, чем шаблонов. Позднее в этом приложении мы остановимся на шаблонах коллекций, которые включены в состав MFC.

Описание приложения Array

Для того чтобы проиллюстрировать возможности работы одного из классов с массивами, мы включили в этот раздел приложение Array. После краткого обзора его работы вы в следующих разделах создадите его самостоятельно (с нашей, конечно, помощью). После запуска программы вы увидите на экране нечто, похожее на то, что показано на рис. Е.1. В окне выведено содержимое массива. Поскольку начальный размер объекта-массива (экземпляра класса CUIntArray) равен 10 (индексы элементов от 0 до 9), на экране вы увидите соответственно 10 строк. Приложение позволяет добавлять элементы в массив, изменять их значения, удалять элементы и все время наблюдать за результатом.

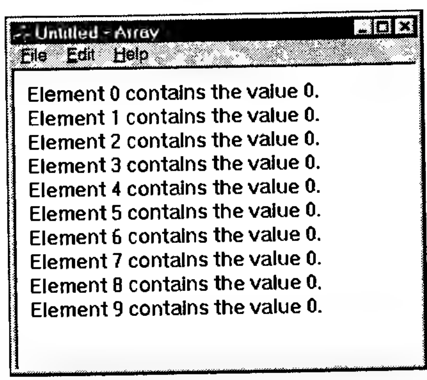


Рис. Е.1. Приложение Array позволяет экспериментировать с классами MFC, обрабатывающими массивы

Элемент в массив можно включить несколькими способами. Чтобы убедиться в этом, щелкните в окне приложения. Появится диалоговое окно, показанное на рис. Е.2. Введите значение индекса в поле Index, а желаемое значение элемента (целое число) — в поле Value. После этого нужно выбрать операцию, которую вы собираетесь выполнить, — вставить элемент, записать значение в существующий элемент или добавить элемент. При выборе переключателя Set значение, которое установлено в поле Value, будет записано в элемент, индекс которого задан полем Index. Выбор переключателя Insert приведет в результате к вставке нового элемента в то место массива, которое задано индексом. При этом все последующие элементы сместятся по направлению к концу массива. И наконец переключатель Add "запустит" операцию добавления элемента — новый элемент будет помещен в "хвост" массива. В этом случае программа ко всем вашим стараниям, сопряженным с вводом значения индекса, отнесется совершенно наплевательски.

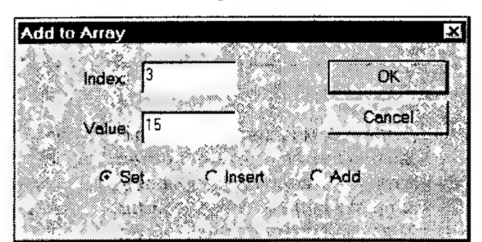


Рис. Е.2. Диалоговое окно Add to Array позволяет добавить элементы в массив

Предположим, например, что вы введете значение 3 в поле Index, а значение 15 — в поле Value и оставите включенным переключатель Set. Щелкните на OK, и результат не заставит себя ждать — он на рис. Е.3. Случилось чудо! Программа установила число 15 в элемент 3 массива, заменив предыдущее значение. А теперь — новый смелый эксперимент. Введите значение 2 в поле Index, а значение 25 (ставки растут!) — в поле Value, выберите переключатель Insert и щелкните на OK. Вы даже не успеете затаить дыхание, как появится результат — в точности как на рис. Е.4. Программа все-таки нашла, куда вставить такое большое число, а остальным пришлось потесниться.

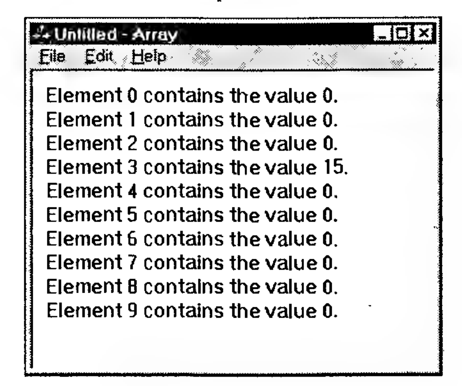


Рис. Е.3. Число 15 записано в элемент массива с индексом 3

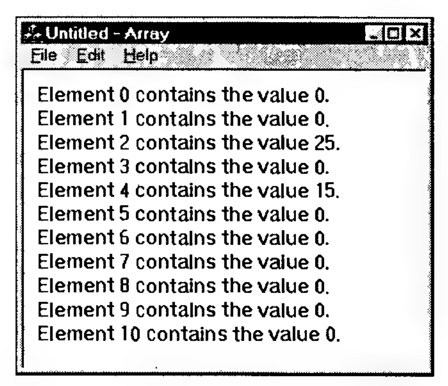


Рис. Е.4. Теперь на экране появился новый элемент в массиве, причем общее число элементов увеличилось до 11

Еще один смелый эксперимент — на сей раз проверим, так ли уж сообразителен класс MFC и так ли уж действительно динамичен создаваемый массив. Задайте значение индекса, выходящее за пределы текущего размера массива. Например, в том интересном положении,

от которого программа еще не оправилась на рис. Е.4, введите значение индекса 20, а значение элемента пусть будет 45. Выберите переключатель Set и щелкните на OK. Результат — на рис. Е.5. Оказалось, что класс все-таки разобрался в ситуации. Хотя элемента с индексом 20 и не существовало, класс сам его создал и установил в нем заданное значение. Теперь можно особенно не беспокоиться о диапазоне значений индексов. Попробовали бы вы позволить себе подобную вольность с прежними С-массивами!

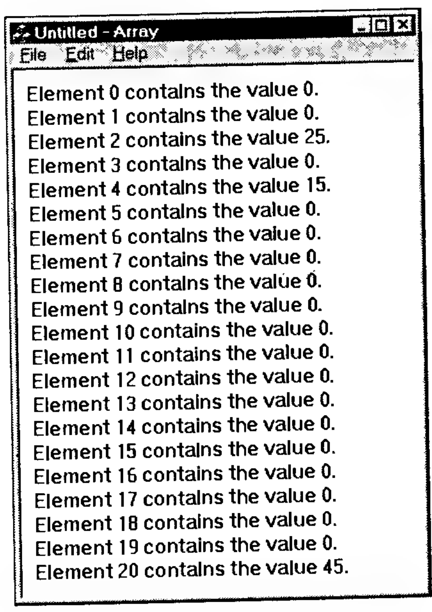


Рис. Е.5. В класс массива добавлены элементы, необходимые для формирования элемента с индексом 20

Совершенно естественно, что, кроме добавления элементов, приложение должно обеспечивать возможность их удаления. Для этого предусмотрено два способа. Чтобы попробовать первый, щелкните на поле окна правой кнопкой мыши. Появится диалоговое окно, показанное на рис. Е.6. Если ввести значение индекса в поле Remove (Удалить) и щелкнуть на OK, программа сотрет указанный элемент массива. Эта операция полностью отменяет результат вставки элемента — элемент удаляется, а массив сжимается. При желании можно установить опцию Remove All (Удалить все). Уж тут-то программа постарается и действительно удалит все элементы массива, но сам объект-массив останется.

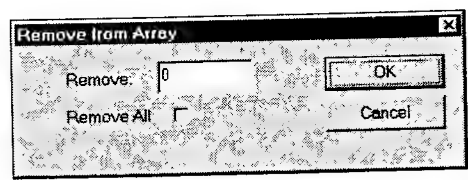


Рис. Е.6. Диалоговое окно Remove from Array позволяет удалять элементы из массива

Объявление и инициализация массива

Теперь, конечно, вы полюбопытствуете, как же программируются все эти фокусы с массивом. Все, оказывается, на удивление просто (как, впрочем, бывает со всяким фокусом). Сначала программа объявляет объект-массив как член данных класса представления:

```
CUIntArray array;
```

Затем в конструкторе класса представления программа инициализирует массив из десяти элементов:

```
array.SetSize(10, 5);
```

Функция SetSize()принимает в качестве аргументов исходное количество элементов и инкремент приращения размера — количество элементов, которые "пристраиваются" к массиву каждый раз, когда для включения нового элемента не хватает существующей размерности. В принципе, обращаться к функции SetSize()совсем необязательно. Однако по умолчанию MFC будет наращивать размер массива порциями по одному элементу, т.е. при каждом включении нового элемента будет происходить обращение к операционной системе за новой порцией памяти, а это замедляет работу программы (хотя, если у вас выполняется достаточно серьезная обработка данных, прежде чем они будут записаны в массив, вы этого замедления и не почувствуете). Если вы не слишком обременяете программу включением новых элементов в массив, поскольку память вам дороже, вообще забудьте о функции SetSize(). Если же, наоборот, вы очень часто включаете в массив новые данные, а размер памяти не играет такой уж важной роли, используйте SetSize() и установите разумное значение инкремента приращения длины массива — это ускорит работу приложения, поскольку реже придется "упрашивать" операционную систему "оказать" программе услугу — выделить "кусочек" памяти.

Добавление элементов в массив

После того как размер массива установлен, программа ожидает, когда же пользователь соизволит щелкнуть кнопкой мыши — все равно какой, левой или правой. Когда это наконец произойдет, программа возьмется за дело — выведет на экран соответствующее диалоговое окно и обработает данные, которые будут в него введены. В листинге Е.1 представлен текст функции OnLButtonDown() приложения Array Demo, которая отрабатывает щелчок левой кнопкой мыши.



В главе 3 описан механизм перехвата щелчка мышью и подробно рассказано, как организовать вызов функции обработки, подобной OnLButtonDown().

Листинг Е.1. Функция CArrayView::OnLButtonDown()

```
void CArrayView::OnLButtonDown(UINT nFlags, CPoint point)
{
   ArrayAddDlg dialog(this);
   dialog.m_index = 0;
   dialog.m_value = 0;
   dialog.m_radio = 0;
   int result = dialog.DoModal();
   if (result == IDOK)
   {
      if (dialog.m_radio == 0)
         array.SetAtGrow(dialog.m_index, dialog.m_value);
```

```
      else if (dialog.m_radio == 1)
         array.InsertAt(dialog.m_index, dialog.m_value, 1);
      else
         array.Add(dialog.m_value);
      Invalidate();
   }
   CView::OnLButtonDown(nFlags, point);
}
```

Первым делом, в функции создается объект и инициализируется диалоговое окно, как это описано в главе 2. Если пользователь завершает манипуляции в окне, щелкнув на кнопке OK, функция OnLButtonDown() анализирует значение переменной m_radio — члена класса диалогового окна. Значение 0 означает, что был выбран первый переключатель (Set), 1 — второй переключатель (Insert), 2 — третий (Add).



В главе 2 обсуждается методика считывания параметров, установленных в диалоговом окне.

Если пользователь пожелал установить значение некоторого элемента массива, программа обращается к функции SetAtGrow() и передает ей в качестве аргумента индекс в массиве и новое значение элемента. В отличие от SetAt(), которая может использовать только индекс, соответствующий текущему размеру массива, функция SetAtGrow() может при необходимости растянуть массив. Это приходится делать в том случае, если заданный индекс превышает максимальный для текущего состояния массива. Именно так произошло, когда вы выбрали установку элемента с индексом 20, а в массиве было всего 11 элементов, т.е. максимальный индекс был равен 10.

Когда пользователь выбирает переключатель Insert, программа вызывает функцию InsertAt() и также передает ей в качестве аргумента индекс в массиве и значение элемента. Это приводит к формированию нового элемента, который вставляется в массив соответственно заданному значению индекса, причем остальные элементы сдвигаются. И наконец, если выбран переключатель Add, программа вызывает функцию Add(), которая добавляет элемент в конец массива. У этой функции единственный аргумент — значение нового элемента. Вызов функции Invalidate() заставляет операционную систему обновить изображение в окне приложения.

Последовательное чтение элементов массива

Для того чтобы пользователь мог увидеть на экране результаты своих манипуляций, функция OnDraw() приложения Array Demo считывает значения элементов массива и выводит их на экран. Текст этой функции представлен в листинге Е.2.



В главе 5 рассматривается методика разработки и вызов функции OnDraw().

Листинг Е.2. Функция CArrayView::OnDraw()

```
void CArrayView::OnDraw(CDC* pDC)
{
   CArrayDoc* pDoc = GetDocument();
   ASSERT_VALID(pDoc);

   // Считывание высоты текущего шрифта.
```

```
    TEXTMETRIC textMetric;
    pDC->GetTextMetrics(&textMetric);
    int fontHeight = textMetric.tmHeignt;
    // Считывание размера массива.
    int count = array.GetSize();
    int displayPos = 10;
    // Вывод на экран данных из массива.
    for (int x=0; x<count; ++x)
    {
        UINT value = array.GetAt(x);
        cnar s[81];
        wsprintf(s, "Element %d contains tne value %u.", x, value);
        pDC->TextOut(10, displayPos, s);
        displayPos += fontHeignt;
    }
}
```

В этой функции сначала считывается высота текущего шрифта, чтобы определить интервал между строками вывода элементов массива на экране. Затем считывается количество элементов в массиве, для чего вызывается функция GetSize(). И наконец программа использует счетчик элементов для организации цикла for, в котором с помощью функции GetAt() считываются значения элементов массива. При этом индекс — переменная цикла и одновременно аргумент вызова GetAt() — заставляет в каждом новом цикле обращаться к следующему элементу. Для вывода на экран программа преобразует числовое значение в текст.

Удаление элементов из массива

Поскольку приложение задумано таким образом, что диалоговое окно удаления вызывается после щелчка правой кнопкой мыши, именно с функции OnRButtonDown() и начинается процедура удаления. Текст этой функции представлен в листинге Е.3.

Листинг Е.3. Функция CArrayView::OnRButtonDown()

```
void CArrayView::OnRButtonDown(UINT nFlags, CPoint point)
{
    ArrayRemoveDlg dialog(this);
    dialog.m_remove = 0;
    dialog.m_removeAll = FALSE;
    int result = dialog.DoModal();
    if (result == IDOK)
    {
        if (dialog.m_removeAll)
            array.RemoveAll();
        else
            array.RemoveAt(dialog.m_remove);
        Invalidate();
    }

    CView::OnRButtonDown(nFlags, point);
}
```

В этой функции после вывода на экран диалогового окна анализируется значение переменной m_removeAll — члена класса диалогового окна. Значение TRUE свидетельствует о том, что пользователь установил соответствующий флажок в диалоговом окне и желает удалить все элементы массива. В этом случае программа вызывает функцию RemoveAll() — член класса массива. Иначе вызывается функция RemoveAt(), единственный аргумент которой — индекс удаляемого элемента. Вызов функции Invalidate()заставляет операционную систему обновить изображение в окне приложения после проведенных манипуляций.

Классы списков

В общем, списки — это несколько своеобразные, я бы сказал, "изощренные" массивы. В классах списков MFC реализован один из видов списков — *связные списки* (linked lists). В таких списках для связей между последовательными элементами (для элементов списков часто используется термин *узел — node*) применяются указатели. Это отличает списки от *чистых* массивов, в которых элементы просто последовательно располагаются в памяти. Списки более пригодны для организации таких множественных структур данных, в которых приходится часто добавлять или исключать элементы. Эти операции в связных списках выполняются гораздо быстрее, причем разница в скорости особенно заметна при большом количестве элементов. Однако поиск элемента в списке протекает медленнее, чем в массиве, поскольку приходится последовательно двигаться по всем элементам, руководствуясь связями-указателями.

Прежде чем приступить к рассмотрению операций со списками, придется несколько пополнить свой словарь терминов. *Голова* списка (head) — это первый узел списка, а *хвост* (tail) — последний узел (рис. Е.7).

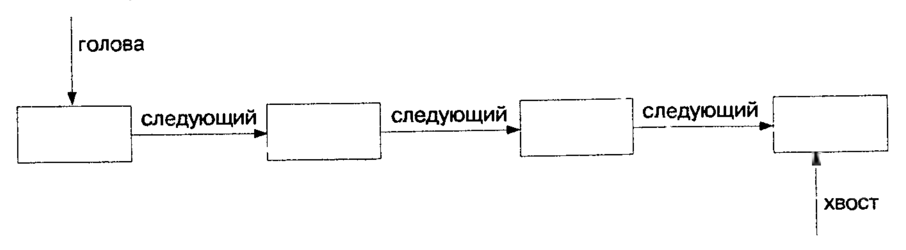


Рис. Е.7. Связный список имеет голову, хвост и множество узлов между ними

Библиотека MFC располагает тремя классами для работы со списками: `CObList` (элементами множества в нем являются объекты), `CPtList` (элементами являются указатели) и `CStringList` (элементами являются текстовые строки — объекты класса `CString`). Все классы списков имеют аналогичные функции-члены, а различаются, в основном, только типом данных, которые они хранят в узлах. В табл. Е.2 представлены функции — члены классов списков — и дано их описание.

Таблица Е.2. Функции-члены классов списков

Функция	Назначение
`AddHead()`	Добавляет узел к голове списка, причем новый узел становится головой
`AddTail()`	Добавляет узел вслед за текущим хвостом списка, причем новый узел становится хвостом
`Find()`	Последовательно просматривает список в поиске объекта. Возвращает значение типа `POSITION`
`FindIndex()`	Последовательно просматривает список в поиске объекта с заданным порядковым номером (индексом). Возвращает значение типа `POSITION` для найденного узла
`GetAt()`	Считывает элемент, соответствующий заданной позиции
`GetCount()`	Считывает количество узлов в списке
`GetHead()`	Считывает головной узел списка
`GetHeadPosition()`	Считывает позицию головного узла списка
`GetNext()`	Считывает очередной узел списка при последовательном его обходе (iterating)
`GetPrev()`	Считывает предшествующий узел списка при последовательном его обходе в обратном порядке

Функция	Назначение
GetTail()	Считывает хвостовой узел списка
GetTailPosition()	Считывает позицию хвостового узла списка
InsertAfter()	Вставляет новый узел вслед за специфицированной позицией в списке
InsertBefore()	Вставляет новый узел перед специфицированной позицией в списке
IsEmpty()	Возвращает TRUE, если список пуст, и FALSE — в противном случае
RemoveAll()	Удаляет все узлы списка
RemoveAt()	Удаляет один узел из списка
RemoveHead()	Удаляет головной узел из списка
RemoveTail()	Удаляет хвостовой узел из списка
SetAt()	Устанавливает узел в специфицированную позицию

Шаблоны списков

Связанные списки — это еще один хороший пример программной конструкции, в которой удобно использовать шаблоны. В библиотеке Standard Template Library, о которой шла речь в главе 26, существуют шаблоны списка и очереди. Но следует прислушаться к мнению разработчиков о том, что использовать классы списков MFC значительно проще, чем шаблоны. Позднее в этом приложении мы остановимся кратко на шаблонах коллекций, которые включены в состав MFC.

Описание приложения List

А теперь, когда вы познакомились с классами списков и их функциями-членами, можно посмотреть, как все это выглядит на практике в работающей программе. Для экспериментов мы выбрали приложение List. Когда вы его запустите, на экране появится окно, показанное на рис. Е.8. В него выведены параметры единственного узла, с которого начинается список. Каждый узел списка может хранить два параметра — оба типа целое.

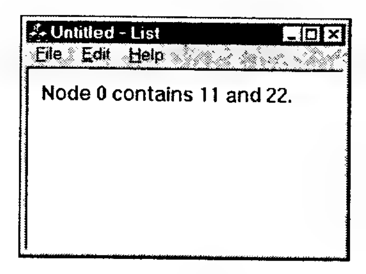


Рис. Е.8. Приложение List в исходном состоянии содержит список, в котором имеется только один узел

Экспериментируя с приложением List, можно будет добавлять новые узлы или удалять существующие из списка. Для включения нового узла щелкните левой кнопкой мыши в окне приложения. Появится диалоговое окно Add Node, показанное на рис. Е.9. Введите в соответствующие поля значения двух параметров, которые будет хранить этот узел, и щелкните на OK. Как только вы проделаете эту несложную операцию, программа присоединит новый узел к хвосту списка и выведет в окно новое содержимое списка. Например, если в диалоговое окно были введены значения 55 и 65, то содержимое списка будет выглядеть так, как показано на рис. Е.10.

Можно также удалить узел из списка. Для этого щелкните правой кнопкой мыши, чтобы на экране появилось диалоговое окно Remove Node (рис. Е.11). Оно предоставляет пользователю не очень богатые возможности: ему придется выбрать, какой узел списка удалить — хвостовой или головной; третьего, к сожалению, не дано. Когда вы закроете это окно, щелкнув на OK, программа удалит заказанный узел и выведет на экран новое состояние списка.

На заметку Если вы попытаетесь удалить узел из пустого списка, приложение выведет на экран окно с предупреждением. Если приложение не будет анализировать такую ситуацию, это может привести к аварийному завершению программы, которая попытается удалить несуществующий узел.

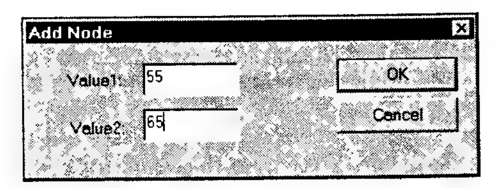


Рис. Е.9. Щелчок левой кнопкой мыши приводит к появлению на экране диалогового окна Add Node

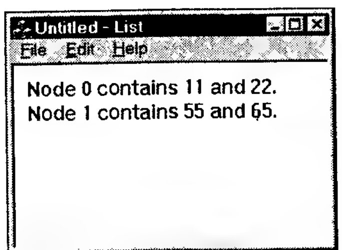


Рис. Е.10. Каждый узел списка хранит два числа

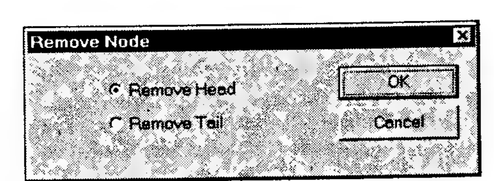


Рис. Е.11. Щелчок правой кнопкой мыши приводит к появлению на экране диалогового окна Remove Node*, которое позволяет удалить узел списка*

Объявление и инициализация списка

Объявление списка ничем не отличается от объявления любого другого типа данных. Просто включите имя класса, который собираетесь использовать, а за ним — идентификатор объекта. Например, в приложении `List` это объявление выглядит следующим образом:

```
CPtrList list;
```

Программа объявляет объект класса `CPtrList`. Этот класс содержит связный список указателей. Такой список может указывать на данные любого типа.

Хотя для инициализации пустого списка большого ума не надо, вам все-таки придется решить, на какой тип данных будут указывать указатели в узлах списка, т.е. нужно объявить структуру узла. Объявление этой структуры в приложении `List` выглядит, как представлено в листинге Е.4.

Листинг Е.4. Структура `CNode`

```
struct CNode
{
        int value1;
        int value2;
};
```

Здесь узел объявлен как структура, содержащая два члена типа `int`. Но вы при желании можете создать структуру с любым набором членов. Для того чтобы включить узел в список, нужно воспользоваться оператором `new`, который обеспечит требуемое количество памяти

для нового экземпляра структуры, а затем добавит указатель на этот экземпляр в список. Приложение List в начальном состоянии содержит список, в котором имеется единственный узел. Он формируется с помощью конструктора класса представления, как это представлено в листинге Е.5.

Листинг Е.5. Конструктор класса CMyListView

```
CMyListView::CMyListView()
{
        CNode* pNode = new CNode;
        pNode->value1 = 11;
        pNode->value2 = 22;
        list.AddTail(pNode);
}
```

Фрагмент текста программы, представленный в этом листинге, сначала формирует новый экземпляр структуры CNode, а затем устанавливает значения ее членов. Метод AddTail() класса списка добавляет узел в хвост списка. Поскольку в этот момент список пуст, то ему, в общем-то, все равно, куда будет добавлен новый узел — в голову или в хвост, т.е. с тем же успехом можно было обратиться и к методу AddHead(). В результате формируется список, в котором всего один узел — он же головной, он же и хвостовой.

Добавление узлов в список

Хотя, в принципе, узлы можно вставлять в любое место списка, самое простое — присоединять их либо к голове, либо к хвосту списка. В результате новый узел становится либо головным в списке, либо хвостовым. В приложении List диалоговое окно **Add Node** выводится на экран после щелчка левой кнопкой мыши, так что разобраться в происходящем вам поможет анализ функции OnLButtonDown(), текст которой представлен в листинге Е.6.

Листинг Е.6. Функция CMyListView::OnLButtonDown()

```
void CMyListView::OnLButtonDown(UINT nFlags, CPoint point)
{
   // Сформировать и инициализировать диалоговое окно.
   AddNodeDlg dialog;
   dialog.m_value1 = 0;
   dialog.m_value2 = 0;
   // Вывести на экран диалоговое окно.
   int result = dialog.DoModal();
   // Если пользователь щелкнул на кнопке OK...
   if (result == IDOK)
   {
      // Сформировать и инициализировать новый узел.
      CNode* pNode = new CNode;
      pNode->value1 = dialog.m_value1;
      pNode->value2 = dialog.m_value2;
      // Вставить узел в список.
      list.AddTail(pNode);
      // Перерисовать окно.
      Invalidate();
   }
   CView::OnLButtonDown(nFlags, point);
}
```

Здесь после вывода на экран диалогового окна программа ждет, когда же пользователь закроет его, щелкнув на кнопке OK. Если это произошло, программа создает и инициализирует

новый узел точно так, как первый узел в конструкторе класса представления. Включение нового узла в список, как и в первом случае, производится функцией AddTail(). Если у вас есть желание разнообразить процедуру включения, подумайте, как предоставить пользователю возможность выбора между включением узла в хвост списка и в голову списка.

Удаление узла из списка

Процедура удаления может быть простой или несколько усложненной — все зависит от того, как выбирать удаляемый узел. Так же, как и при включении узла, операция с произвольным узлом списка потребует, в первую очередь, локализовать искомый узел и получить его позицию в списке. О позиции узла мы поговорим в следующем разделе, в котором речь пойдет о последовательном обходе узлов. А здесь для простоты программа позволяет пользователю выбрать один из двух вариантов — удалить головной узел или хвостовой. Соответствующий фрагмент программы представлен в листинге Е.7.

Листинг Е.7. Функция CMyListView::OnRButtonDown()

```
void CMyListView::OnRButtonDown(UINT nFlags, CPoint point)
{
   // Сформировать и инициализировать диалоговое окно.
   RemoveNodeDlg dialog;
   dialog.m_radio = 0;
   // Вывести на экран диалоговое окно.
   int result = dialog.DoModal();
   // Если пользователь щелкнул на кнопке OK...
   if (result == IDOK)
   {
      CNode* pNode;
      // Проверить, не пустой ли список.
      if (list.IsEmpty())
         MessageBox("No nodes to delete.");
      else
      {
         // Удалить заданный узел.
         if (dialog.m_radio == 0)
            pNode = (CNode*)list.RemoveHead();
         else
            pNode = (CNode*)list.RemoveTail();
         // Уничтожить объект узла и перерисовать окно.
         delete pNode;
         Invalidate();
      }
   }
   CView::OnRButtonDown(nFlags, point);
}
```

Как и в предыдущем случае, после вывода на экран диалогового окна программа ждет, когда же пользователь закроет его, щелкнув на кнопке OK. А после этого анализируется, какой выбор сделал пользователь — удалять хвостовой или головной узел. Если был выбран переключатель Remove Head (Удалить в голове), то переменная m_radio — член класса диалогового окна — будет равна 0. Соответственно программа обратится к методу RemoveHead() класса списка. В противном случае будет вызвана функция RemoveTail(). Обе функции возвращают указатель на удаленный узел. Но прежде, чем приступать к экзекуции списка, нужно удостовериться, что объект экзекуции на месте. Обратите внимание на то, что перед анализом состояния переключателей диалогового окна в программе стоит вызов функции IsEmpty()и анализ возвращаемого ею значения. Если в списке нет узлов, экзекуция отменяется.

На заметку Обратите внимание на еще одну деталь. Чтобы освободить память, выделенную для узла, программа использует оператор delete, которому передается указатель, возвращенный вызванным методом удаления узла. Нужно помнить, что само по себе удаление узла из списка не уничтожает его как экземпляр класса CNode — из списка удаляется только указатель на этот объект. Очистить память от объекта можно только с помощью оператора delete.

Последовательный обход узлов списка

Часто возникает необходимость "пройтись" (iterate over) по всем узлам списка. Например, в приложении List такой обход нужен для того, чтобы вывести на экран содержимое списка — от головы до хвоста. Точнее, такой обход выполняется в функции OnDraw(), текст которой представлен в листинге Е.8.

Листинг Е.8. Функция CMyListView::OnDraw()

```
void CMyListView::OnDraw(CDC* pDC)
{
   CListDoc* pDoc = GetDocument();
   ASSERT_VALID(pDoc);
   // Считать высоту текущего шрифта.
   TEXTMETRIC textMetric;
   pDC->GetTextMetrics(&textMetric);
   int fontHeignt = textMetric.tmHeight;
   // Инициировать переменные, используемые для организации цикла.
   POSITION pos = list.GetHeadPosition();
   int displayPosition = 10;
   int index = 0;
   // Просмотреть список, выводя на экран содержимое каждого узла.
   while (pos != NULL)
   {
      CNode* pNode = (CNode*)list.GetNext(pos);
      char s[81];
      wsprintf(s, "Node %d contains %d and %d.",
         index, pNode->value1, pNode->value2);
      pDC->TextOut(10, displayPosition, s);
      displayPosition += fontHeignt;
      ++index;
   }
}
```

Позиция головного узла списка возвращается функцией GetHeadPosition() — членом класса списка. *Позиция* — это параметр, который используется во многих методах классов списков для быстрого доступа к специфицированному узлу. Нам нужна позиция начального узла как отправная точка для обхода всего списка.

Сам по себе обход организован в виде цикла while. В каждом очередном цикле вызывается функция GetNext(), единственным аргументом которой является ссылка на позицию текущего узла. Эта функция возвращает указатель на очередной узел и одновременно устанавливает позицию следующего узла переменной pos. Когда программа доходит до последнего узла, в переменной pos оказывается значение NULL. В приведенном тексте программы именно условие (pos != NULL) выбрано в качестве условия продолжения цикла.

Удаление списка

Еще один случай, где не обойтись без обхода узлов списка, — это его удаление. Такая необходимость возникает при завершении работы со списком или при завершении работы приложения, когда нужно освободить память, занятую узлами списка, на которые указывают ука-

затели. В приложении List эта задача возложена на деструктор класса списка, текст которого приведен в листинге Е.9.

Листинг Е.9. Деструктор класса CMyListView

```
CMyListView::~CMyListView()
{
   // Пройти по всем узлам, последовательно удаляя каждый.
   while (!list.IsEmpty())
   {
      CNode* pNode = (CNode*)list.RemoveHead();
      delete pNode;
   }
}
```

В деструкторе обход также происходит в цикле while, который выполняется до тех пор, пока функция IsEmpty()не возвратит TRUE. В теле цикла программа удаляет головной узел из списка, в результате чего очередной узел становится головным и стирает его, освобождая выделенную для узла память. Когда таким образом будут удалены все узлы, память, занятая ими, полностью освободится.

Внимание!

Не забудьте, что любой объект, созданный с помощью оператора new, обязательно нужно удалить. Если вы не сделаете этого, появится "утечка" памяти. Конечно, в маленькой программе потеря нескольких байтов может вообще оказаться незамеченной, но, если программа работает достаточно долго, создавая и удаляя сотни и тысячи объектов, подобных узлам списков, вы можете столкнуться с проблемой нехватки памяти в самый неожиданный момент. Так что запомните: убирать за собой — это не только дань вежливости, но и жизненная необходимость для любого программиста.



В главе 24 проблемы, связанные с распределением и утилизацией системной памяти, рассматриваются более подробно.

Классы ассоциированных списков

Для создания таблиц просмотра содержимого коллекций MFC предоставляет в распоряжение программиста классы ассоциированных списков. Например, при необходимости можно преобразовать с их помощью цифры в слова, представляющие соответствующие числа. Таким образом, можно использовать цифру 1 в качестве ключа для поиска слова один. Для решения такого рода задач классы ассоциированных списков (mapped classes) являются идеальным инструментом. Именно благодаря наличию в MFC таких классов можно использовать множество типов данных в качестве ключей и значений.

Набор классов ассоциированных списков в MFC включает CMapPtrToPtr, CMapPtrToWord, CMapStringToOb, CMapStringToPtr, CMapStringToString, CMapWordToOb и CMapWordToPtr. Первый тип данных в имени класса — это тип данных ключа, а второй — тип данных значения. Например, CMapStringToOb использует строковую переменную как ключ, а объект — как значение; CMapStringToString — класс, который мы будем рассматривать в учебном приложении этого раздела, — использует строковую переменную и в качестве ключа, и в качестве значения. Все классы ассоциированных списков построены по одному принципу и имеют совершенно аналогичные методы, перечисленные в табл. Е.3.

Таблица Е.3. Методы классов ассоциированных списков

Функция	Назначение
GetCount()	Считывает количество элементов карты ассоциации
GetNextAssoc()	Считывает следующий элемент при последовательном переборе элементов карты ассоциации
GetStartPosition()	Считывает позицию первого элемента карты ассоциации
IsEmpty()	Возвращает TRUE, если карта ассоциации пуста, и FALSE — в противном случае
Lookup()	Выполняет поиск значения, ассоциированного с ключом
RemoveAll()	Удаляет все элементы карты ассоциации
RemoveKey()	Удаляет один элемент из карты ассоциации
SetAt()	Добавляет элемент в карту ассоциации или заменяет элемент, ключ которого соответствует заданному

Описание приложения Map

Учебное приложение Map, которое рассматривается в этом разделе, выводит содержимое карты ассоциации и позволяет пользователю извлекать значения из карты, задавая определенный ключ. Непосредственно после запуска программа выводит окно, показанное на рис. Е.12.

В окне представлено содержимое объекта класса ассоциированного списка. Здесь цифры являются ключами, определяющими доступ к словам, которые в "словесной" форме представляют соответствующие числа. Для того чтобы вытащить из карты (объекта класса) некоторое значение, щелкните на поле окна мышью. Появится диалоговое окно, показанное на рис. Е.13. Введите значение ключа в поле Key и щелкните на OK. Программа найдет соответствующую ключу величину — слово-число — и выведет его в другом окне сообщения (рис. Е.14). Введите значение ключа в поле Key и щелкните на OK. Программа найдет соответствующую ключу величину — слово-число — и выведет его в другом окне сообщения (рис. Е.14).

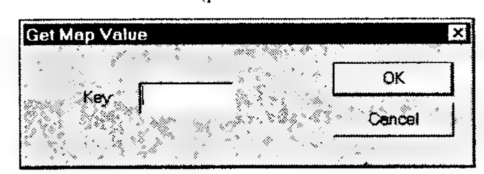


Рис. Е.13. Диалоговое окно Get Map Value позволяет выполнять поиск соответствия между заданным значением ключа и величинами, хранящимися в карте отображения

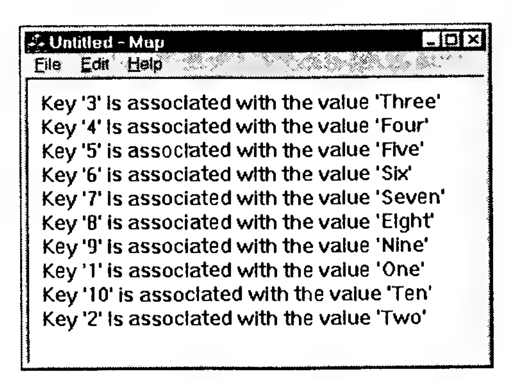


Рис. Е.12. Приложение Map выводит содержимое карты ассоциации — объекта класса

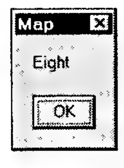


Рис. Е.14. В этом окне сообщения выводится соответствующая величина, найденная в карте отображения

Объявление и инициализация карты ассоциации

В исходном состоянии карта ассоциации приложения Map состоит из десяти элементов. Объект класса ассоциированного списка объявляется как член данных класса представления:

```
CMapStringToString map;
```

Это объект класса CMapStringToString, в котором используется строковая переменная и в качестве ключа, и в качестве значения.

Само по себе объявление объекта класса не заполняет его данными. Сделать это нужно отдельно, что и выполняется в конструкторе класса представления приложения Map. Соответствующий фрагмент программы представлен в листинге Е.10.

Листинг Е.10. Конструктор класса CMapView

```
CMapView::CMapView()
{
        map.SetAt("1", "One");
        map.SetAt("2", "Two");
        map.SetAt("3", "Three");
        map.SetAt("4", "Four");
        map.SetAt("5", "Five");
        map.SetAt("6", "Six");
        map.SetAt("7", "Seven");
        map.SetAt("8", "Eight");
        map.SetAt("9", "Nine");
        map.SetAt("10", "Ten");
}
```

Функция SetAt() — член класса CMapStringToString — принимает в качестве аргументов ключ и ассоциированное с ним значение. Если такой ключ уже имеется в карте отображения, функция заменяет ассоциированное с ним значение тем значением, которое передано в качестве второго аргумента.

Извлечение значения из карты ассоциации

Диалоговое окно Get Map Value появляется на экране после щелчка на поле окна приложения. Так что нет ничего удивительного в том, что анализ программы мы начнем с функции OnLButtonDown() — члена класса представления. Ее текст представлен в листинге Е.11.

Листинг Е.11. Функция CMapView::OnLButtonDown()

```
void CMapView::OnLButtonDown(UINT nFlags, CPoint point)
{
   // Сформировать и инициализировать диалоговое окно.
   GetMapDlg dialog(this);
   dialog.m_key = "";
   // Вывести на экран диалоговое окно.
   int result = dialog.DoModal();
   // Если пользователь щелкнул на кнопке OK...
   if (result == IDOK)
   {
      // Просмотр запрошенных данных.
      CString value;
      BOOL found = map.Lookup(dialog.m_key, value);
      if (found)
         MessageBox(value);
```

```
      else
         MessageBox("No matching value.");
   }
   CView::OnLButtonDown(nFlags, point);
}
```

В функции OnLButtonDown(), как мы не раз видели в этой главе, организуется вывод на экран диалогового окна и ожидается, пока пользователь завершит с ним работу и щелкнет на OK. Затем вызывается функция Lookup()класса ассоциированного списка, которая использует введенное значение ключа в качестве первого аргумента. Второй аргумент — это ссылка на строку-приемник, в которую функция перешлет найденное значение, ассоциированное с ключом. Если ключ не будет найден в объекте ассоциации, функция вернет FALSE, в противном случае — TRUE. Вызывающая программа использует возвращаемое значение с единственной целью — чтобы решить, что выводить на экран: содержимое строки-приемника, заполненное Lookup(), или окно предупреждающего сообщения.

Последовательный перебор элементов карты ассоциации

Для того чтобы вывести на экран содержимое объекта класса ассоциированного списка, программа должна последовательно перебрать все элементы карты ассоциации. Так же, как и в приложениях, демонстрирующих возможности массивов и списков, эта задача в приложении Map возлагается на функцию OnDraw(), текст которой представлен в листинге Е.12.

Листинг Е.12. Функция CMapView::OnDraw()

```
void CMapView::OnDraw(CDC* pDC)
{
   CMapDoc* pDoc = GetDocument();
   ASSERT_VALID(pDoc);
   TEXTMETRIC textMetric;
   pDC->GetTextMetrics(&textMetric);
   int fontHeight = textMetric.tmHeight;
   int displayPosition = 10;
   POSITION pos = map.GetStartPosition();
   CString key;
   CString value;
   while (pos != NULL)
   {
      map.GetNextAssoc(pos, key, value);
      CString str = "Key " + key +
         " is associated with the value " +
         value + "";
      pDC->TextOut(10, displayPosition, str);
      displayPosition += fontHeight;
   }
}
```

Многое в этой версии очень напоминает другие версии, которые уже анализировались нами ранее в этой главе. Начинается процесс с вызова функции GetStartPosition() — члена класса ассоциированного списка, которая возвращает позицию первого элемента карты ассоциации (это может быть совсем не тот элемент, который был первым по времени вставлен в карту). Сам по себе обход организован в виде цикла while. В каждом очередном цикле вызывается функция GetNextAssoc(), одним из аргументов которой является ссылка на позицию текущего элемента — локальную переменную pos. В первом цикле используется значение позиции, возвращенное функцией GetStartPosition(). Функция же GetNextAssoc() извлекает

ключ и значение из текущего элемента карты и переставляет позицию на следующий элемент. Когда программа доходит до последнего элемента узла, в переменной pos оказывается NULL и цикл обхода завершается.

Шаблоны класса коллекций

В библиотеку MFC включены шаблоны класса, с помощью которых можно создавать собственные специальные классы коллекций. Более подробная информация о шаблонах содержится в соответствующем разделе главы 26. Хотя сама по себе тема *шаблоны в C++* достаточно объемна, использование шаблонов в этом конкретном случае — по отношению к классам коллекций — не вызывает особых затруднений. Предположим, вы хотите сформировать класс массива, который смог бы включать в качестве элементов массива структуры, приведенные в листинге Е.13.

Листинг Е.13. Пример структуры

```
struct MyValues
{
        int value1;
        int value2;
        int value3;
};
```

Сначала для создания нового класса необходимо использовать шаблон:

```
CArray<MyValues, MyValues&> myValueArray;
```

Здесь CArray — это шаблон, который используется для создания собственного класса массива. Два аргумента шаблона — тип данных, который нужно хранить в массиве, и тип данных, который функции-члены нового класса массива должны использовать в качестве аргумента там, где это необходимо. В нашем случае тип данных, хранящнхся в массиве, — это структуры MyValues. Второй же аргумент шаблона указывает, что функции-члены нового класса массива должны использовать в качестве аргумента ссылку на структуру типа MyValues.

Для формирования самого массива нужно установить его начальный размер:

```
myValueArray.SetSize(10, 5);
```

Теперь можно позволить себе добавлять элементы в этот массив:

```
MyValues myValues;

myValueArray.Add( myValues);
```

Поскольку новый класс массива сформирован по шаблону, этот массив можно использовать так же, как любой объект класса массивов MFC (что уже было опнсано выше). Другие шаблоны класса коллекцнй MFC, которые можно спокойно использовать в своих разработках, — это CList и CMap. Что же из всего сказанного следует? А следует то, что в вашем распоряжении, когда вы создаете собственные классы подобного тнпа, теперь оказываются результаты той гнгантской работы, которую проделали программисты, создававшие MFC. Вы можете построить массив объектов класса Employee (Служащие) или связный список из объектов класса Order (Заказы) или карту ассоциации, в которой связываются имена с объектами класса Customer (Покупатели).

Класс CString

Едва ли найдется программа, в которой не обрабатывались бы тем или иным образом текстовые строки. К сожалению, сам по себе язык C++ не имеет мощного встроенного механизма работы с данными такого типа, что позволяет некоторым программистам говорить о его "недоразвитости" по сравнению с PASCAL или BASIC. Встроенный в MFC класс CString переводит подобные рассуждения в разряд злопыхательских инсинуаций. Методы этого класса ни в чем не уступают аналогичным средствам других языков. В табл. Е.4 перечислены чаще всего употребляемые методы этого класса.

Таблица Е.4. Чаще всего используемые методы класса CString

Функция	Назначение
Compare()	Сравнивает две строки с учетом регистра символов
CompareNoCase()	Сравнивает две строки, игнорируя регистры символов
Empty()	Очищает строку
Find()	Находит подстроку
Format()	Преобразует данные других типов в текст, подобно стандартной функции sprintf
GetAt()	Считывает символ, который находится в заданной позиции в строке
GetBuffer()	Возвращает указатель на содержимое строки
GetLength()	Возвращает количество символов в строке
IsEmpty()	Возвращает TRUE, если в строке нет ни одного символа
Left()	Возвращает левый сегмент строки (предшествующий заданной позиции)
MakeLower()	Преобразует все символы строки в строчные (нижнего регистра)
MakeReverse()	"Переворачивает" строку — меняет порядок символов на обратный
MakeUpper()	Преобразует все символы строки в прописные (верхнего регистра)
Mid()	Возвращает фрагмент строки, "выдернутый" из середины (находящийся между двумя заданными позициями)
Right()	Возвращает правый сегмент строки (следующий за заданной позицией)
SetAt()	Устанавливает символ в специфицированную позицию
TrimLeft()	Удаляет ведущие пробелы из строки
TrimRight()	Удаляет хвостовые пробелы из строки

Помимо перечисленных функций, в классе CString определен полный набор терминальных операторов для работы с текстовыми строками. С их помощью можно выполнять *конкатенацию* (т.е. объединение, слияние) строк. Оператор (+) присваивает строке новое значение, оператор (=) выполняет преобразование машинного формата строки (например, получает так называемый *С-стиль*), используя оператор LPCTSTR, и т.п.

Формирование объекта класса CString выполняется стандартным для языка C++ способом:

```
CString str = "This is test string";
```

Конечно, существует много способов формирования подобного объекта. Наш пример демонстрирует только один из возможных вариантов. Можно создать пустой объект и позже присвоить ему некоторое значение либо создать объект, копируя существующий или заказав n-кратное повторение некоторого символа.

Когда экземпляр класса создан, можно вызывать методы класса и манипулировать строкой, как вам вздумается. Например, можно преобразовать все символы строки в прописные (символы верхнего регистра):

```
str.MakeUpper();
```

Для того чтобы удлинить строку, можно воспользоваться переопределенными терминальными операторами (+) и (+=):

```
CString sentence = "hello " + str;

sentence += "there.";
```

Для сравнения двух строк нужно вызвать функцию-член класса:

```
str.Compare("Test String");
```

Можно также сравнивать два экземпляра класса CString:

```
CString testStr = "Test String";

str.Compare( testStr);
```

А самый изящный вариант таков:

```
if( testStr == str)
```

Если вы полистаете оперативную документацию, то найдете массу примеров использования самых разнообразных методов класса CString. Можете поверить мне на слово, что использовать эти методы не сложнее, чем применять встроенные функции BASIC или PASCAL.

Классы для работы со временем

Если вам приходилось программировать обработку данных, связанных с текущим временем, которые можно получить с компьютера, вас, несомненно, порадует наличие в MFC таких классов, как CTime и CTimeSpan. Они представляют соответственно абсолютное и относительное (elapsed) время. Использование этих классов настолько очевидно, что мы даже отказались от специального примера в данном разделе. Тем не менее определенные начальные сведения об этих полезных классах вы из него почерпнете. Прежде чем приступить к самостоятельным экспериментам с классами времени, взгляните на табл. E.5, в которой перечислены методы класса Ctime, и на аналогичную таблицу для класса CTimeSpan (табл. E.6).

Таблица E.5. Функции-члены класса CTime

Функция	Назначение
Format()	Создает текстовую строку, представляющую время соответственно содержимому объекта
FormatGmt()	Создает текстовую строку, представляющую время по Гринвичу (в формате GMT — Greenwich mean time) соответственно содержимому объекта
GetCurrentTime()	Заполняет объект класса CTime соответственно текущему времени
GetDay()	Считывает текущий день (как целое число) соответственно содержимому объекта
GetDayOfWeek()	Считывает текущий день недели соответственно содержимому объекта, начиная с 1 для воскресенья
GetGmtTm()	Считывает содержимое объекта в структуру tm, раскладывая его на компоненты — секунды, минуты, часы, месяц, год, день недели, день в году

Функция	Назначение
GetHour()	Считывает текущий час (как целое число) соответственно содержимому объекта
GetLocalTm()	Считывает содержимое объекта, преобразуя его в местное время, в структуру tm: раскладывает его на компоненты — секунды, минуты, часы, месяц, год, день недели, день в году
GetMinute()	Считывает минуты (как целое число) из содержимого объекта
GetMonth()	Считывает месяц (как целое число) из содержимого объекта
GetSecond()	Считывает секунды (как целое число) из содержимого объекта
GetTime()	Считывает время соответственно содержимому объекта, преобразуя его в тип time_t
GetYear()	Считывает год (как целое число) из содержимого объекта

Таблица Е.6. Функции-члены класса CTimeSpan

Функция	Назначение
Format()	Создает текстовую строку, представляющую время соответственно содержимому объекта
GetDays()	Считывает количество дней соответственно содержимому объекта
GetHours()	Считывает количество часов соответственно содержимому объекта
GetMinutes()	Считывает количество минут соответственно содержимому объекта
GetSeconds()	Считывает количество секунд соответственно содержимому объекта
GetTotalHours()	Считывает суммарное количество часов соответственно содержимому объекта
GetTotalMinutes()	Считывает суммарное количество минут соответственно содержимому объекта
GetTotalSeconds()	Считывает суммарное количество секунд соответственно содержимому объекта

Использование объекта класса CTime

Для создания объекта класса CTime нужно просто вызвать функцию GetCurrentTime():

```
CTime time = CTime::GetCurrentTime();
```

Поскольку GetCurrentTime() является статической функцией класса CTime, ее можно вызывать без предварительного создания экземпляра класса. Но имя класса нужно обязательно включить в оператор вызова. Как видно, функция возвращает объект класса CTime, содержимое которого соответствует текущему времени. Если нужно вывести отсчет времени на экран, придется обратиться к функции Format():

```
CString str = time.Format("DATE: %A, %B %d, %Y");
```

Функция Format() требует в качестве единственного аргумента форматную строку, которая определяет вид выходного текста. Предыдущий пример сформирует такой текст:

```
DATE: Saterday, April 20, 1996
```

Форматная строка для Format() строится примерно по таким же правилам, что и аналогичный аргумент всем известной функции printf() для работы в DOS или функции wsprintf() для работы в Windows. В нее также включаются литералы, знаки препинания, например "DATE:" в нашем примере, и форматирующие символы, которые затем замещаются значениями. Форматирующий символ %A в предыдущем примере заменяется наименованием дня недели, а %B — наименованием месяца. Таким образом, разница состоит только в форматирующих символах. Символы, которые используются в функции Format(), перечислены в табл. Е.7.

Таблица Е.7. Форматирующие символы функции Format()

Код	Описание
%a	Сокращенное наименование дня недели, например Sat вместо Saturday
%A	Полное наименование дня недели
%b	Сокращенное наименование месяца, например Mar вместо March
%B	Полное наименование месяца
%c	Дата и время в формате, соответствующем локализованной версии продукта. Например, для США это будет выглядеть так: 03/17/96 12:15:34
%d	День месяца как число (01-31)
%H	Часы в сутках (00-23)
%I	Часы в 12-часовом формате (01-12)
%j	День в году — число (001-366)
%m	Месяц как число (1-12)
%M	Минуты как число (00-59)
%p	Локализованное представление в 12-часовом формате (A.M./P.M.)
%S	Секунды как число (00-59)
%U	Неделя в году как число (00-51; полагается, что воскресенье — первый день очередной недели)
%w	День недели как число (0-6; полагается, что воскресенье имеет номер 0)
%W	Неделя в году как число (00-51; полагается, что понедельник — первый день очередной недели)
%x	Локализованное представление даты
%X	Локализованное представление времени
%y	Год без первых двух цифр, соответствующих столетию (00-99)
%y	Год без первых двух цифр, соответствующих столетию (00-99)
%Y	Год полностью — с префиксом столетия
%z	Сокращенное имя временной зоны
%Z	Полное имя временной зоны
%%	Знак процента

Другие методы класса Ctime, такие как GetMinute(), GetMonth() и GetYear(), в комментариях не нуждаются. Однако мы все-таки покажем пример использования метода GetLocalTm():

```
struct tm* timeStruct;

timeStruct - time.GetLocalTm();
```

В первом операторе объявляется указатель на структуру tm. Сама структура объявлена в Visual C++, а текст объявления приведен в листинге Е.14. Во втором операторе этому указателю присваивается значение, возвращаемое функцией GetLocalTm(), причем соответствующая структура уже будет заполнена данными.

Листинг Е.14. Структура tm

```
struct tm
{
        int tm_sec;                /* секунды после минут - [0,59] */
        int tm_min;                /* минуты после часа - [0,59] */
        int tm_hour;       /* часы после полуночи - [0,23] */
        int tm_mday;       /* день месяца - [1,31] */
        int tm_mon;                /* месяц - [0,11] */
        int tm_year;       /* год, начиная с 1900 */
        int tm_wday;       /* день недели, начиная с воскресенья - [0,6] */
        int tm_yday;       /* день, начиная с 1-го января - [0,365] */
        int tm_isdst;      /* флаг половины суток */
};
```

На заметку Класс CTime располагает множеством перегруженных конструкторов, которые позволяют создавать экземпляры класса самыми различными способами, используя самые различные варианты представления времени.

Использование объекта класса CTimeSpan

Объект класса CTimeSpan есть не что иное как разница между двумя отсчетами времени. Можно использовать объект класса CTime вместе с объектом класса CTimeSpan, чтобы определить, сколько времени прошло между двумя отсчетами. Сначала нужно создать объект класса CTime для текущего времени. Затем, когда нечто свершится, нужно создать второй объект класса CTime для нового отсчета. Вычитание нового объекта класса из старого CTime породит объект класса CTimeSpan, который и будет содержать данные о временном промежутке между отсчетами. Пример подобной программки приведен в листинге Е.15.

Листинг Е.15. Определение временного интервала

```
CTime startTime = CTime::GetCurrentTime();
//..
//..
//..
CTime endTime = (   ne::GetCurrentTime();
CTimeSpan timeSp   = endTime - startTime;
```

Предметный указатель

Г

Д

Ж

З

И

К

М

Н

О

П

Р

С

T

У

Ф

Х

Ц

Ч

Ш

Э

Я

Оглавление

Учебное пособие

Кэйт Грегори

Использование Visual C++ 6.

Специальное издание.

Оригинал-макет подготовлен отделом компьютерной верстки
издательского дома "Вильямс"

ЗАО «Компьютерное издательство "Диалектика"». 105215, г. Москва,
ул. Константина Федина, д. 1, корп. 1. Изд. лиц. ЛР № 090216 от 7.10.98.

Подписано в печать 26.02.99. Формат 70×100/16. Гарнитура Times. Печать офсетная.
Усл. печ. л. 69,66. Уч.-изд. л. 67,95. Тираж 4000 экз. Заказ № 527.

Отпечатано с диапозитивов в ГПП «Печатный Двор» Государственного комитета РФ по печати.
197110, С.-Петербург, Чкаловский пр., 15.

ЗАМЕТКИ

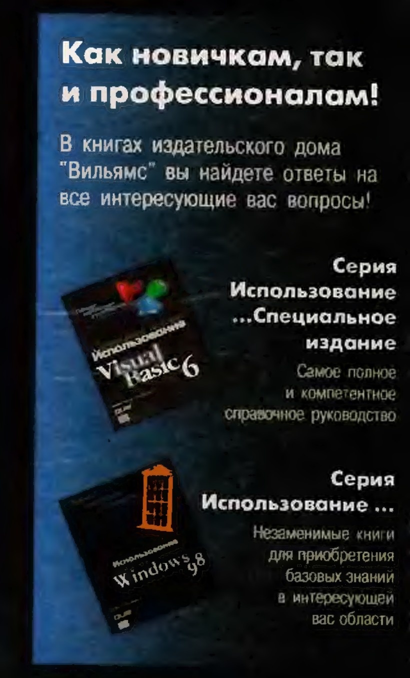
Специальное издание

Самое полное руководство по Visual C++ 6

- Неограниченные возможности программирования в среде Visual C++ 6
- Методика использования Visual Studio при разработке приложений в среде Visual C++
- Технология применения элементов управления Windows в приложениях, в том числе новых средств работы с адресами в Internet, датами и календарем
- Разработка приложений, включающих современные средства интерфейса с пользователем – панели управления, строки состояния, систему оперативной справки и многое другое
- Включение в приложение средств управления печатью документов
- Создание в среде Visual C++ типовых и составных элементов управления ActiveX™
- Использование библиотеки Active Template Library (ATL) в процессе разработки элементов управления ActiveX™
- Использование всех возможностей библиотеки Microsoft® Foundation Classes
- Программирование операций с базами данных с использованием ODBC или технологии Active Data Objects (ADO)
- Создание в среде Visual C++ приложений, работающих в сети Internet
- Разработка надежных программ с использованием новейших технологий параллельного выполнения задач, а также обработки исключений и шаблонов ActiveX

Категория: программирование
Предмет рассмотрения: Visual C++
Уровень: для начинающих и пользователей средней квалификации


ISBN 5-8275-0022-4
99022
9 785827 500223



Кэйт Грегори – соучредительница фирмы *Gregory Consulting Limited*, специализирующейся на обучении пользователей и разработке заказных программных продуктов. Кроме того, Кэйт – редактор журнала *Visual C++ Developer*. Она часто выступает с докладами на ежегодно проводимых компанией Microsoft Днях разработчика. Занимается программированием с 1979 года, свое первое сообщение отослала по электронной почте более двадцати лет назад. Кэйт – автор книг *Building Internet Applications with Visual C++*, *Special Edition Using Visual C++ 4.2* и *Special Edition Using Visual C++ 5*.





www.quecorp.com
www.williams.kiev.ua